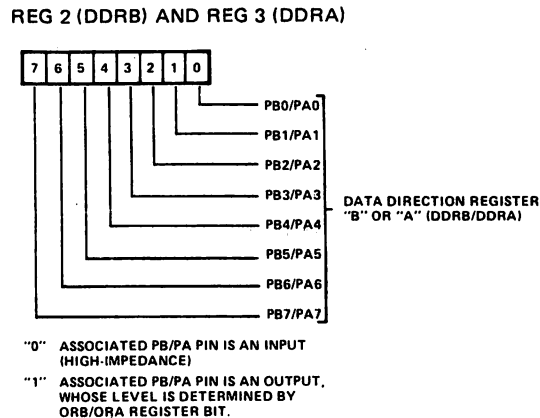


**Figure H-11: Data Direction Registers (DDRB, DDRA)**

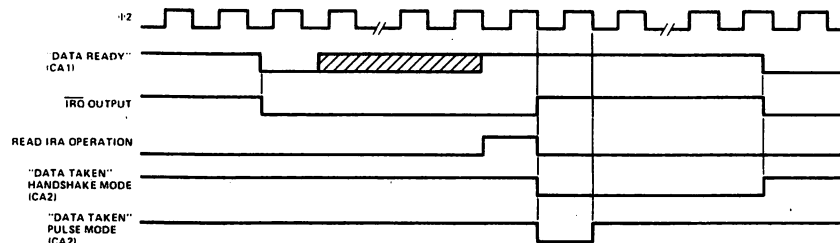


## READ HANDSHAKE

Positive control of data transfers from peripheral devices into the system processor can be accomplished very effectively using Read Handshaking. In this case, the peripheral device must generate the equivalent of a "Data Ready" signal to the processor signifying that valid data is present on the peripheral port. This signal normally interrupts the processor, which then reads the data, causing generation of a "Data Taken" signal. The peripheral device responds by making new data available. This process continues until the data transfer is complete.

In the SY6522, automatic "Read" Handshaking is possible on the Peripheral A port only. The CA1 interrupt input pin accepts the "Data Ready" signal and CA2 generates the "Data Taken" signal. The "Data Ready" signal will set an internal flag which may interrupt the processor or which may be polled under program control. The "Data Taken" signal can either be a pulse or a level which is set low by the system processor and is cleared by the "Data Ready" signal. These options are shown in Figure H-12, which illustrates the normal Read Handshaking sequence.

**Figure H-12: Read Handshake Timing (Port A, Only)**

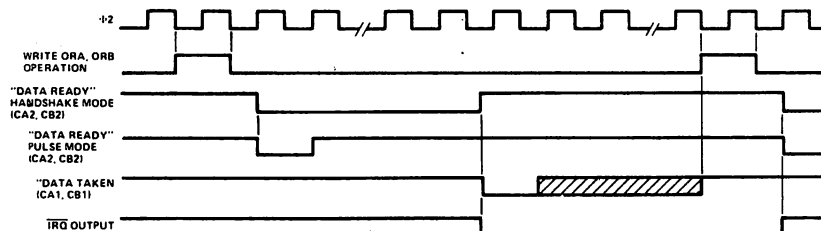


## WRITE HANDSHAKE

The sequence of operations which allows handshaking data from the system processor to a peripheral device is very similar to that described for Read Handshaking. However, for Write Handshaking, the SY6522 generates the "Data Ready" signal and the peripheral device must respond with the "Data Taken" signal. This can be accomplished on both the PA port and the PB port on the SY6522. CA2 or CB2 act as a "Data Ready" output in either the handshake mode or pulse mode and CA1 or CB1 accept the "Data Taken" signal from the peripheral device, setting the Interrupt Flag and cleaning the "Data Ready" output. This sequence is shown in Figure H-13.

Selection of operating modes for CA1, CA2, CB1, and CB2 is accomplished by the Peripheral Control Register (Figure H-14).

**Figure H-13: Write Handshake Timing**



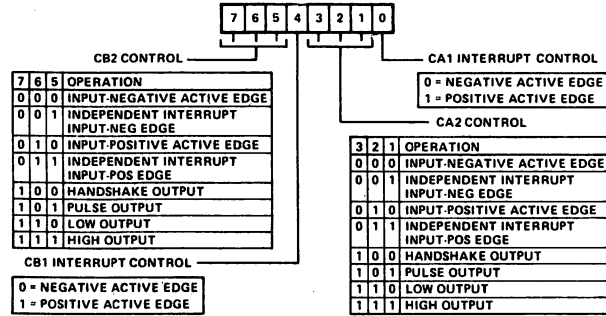
## TIMER OPERATION

Interval Timer T1 consists of two 8-bit latches and a 16-bit counter. The latches are used to store data which is to be loaded into the counter. After loading, the counter decrements at  $\phi 2$  clock rate. Upon reaching zero, an Interrupt Flag will be set, and IRQ will go low if the interrupt is enabled. The timer will then disable any further interrupts, or will automatically transfer the contents of the latches into the counter and will continue to decrement. In addition, the timer may be programmed to invert the output signal on a peripheral pin each time it "times-out". Each of these modes is discussed separately below.

The T1 counter is depicted in Figure H-15 and the latches in Figure H-16.

**Figure H-14: CA1, CA2, CB1, CB2 Control**

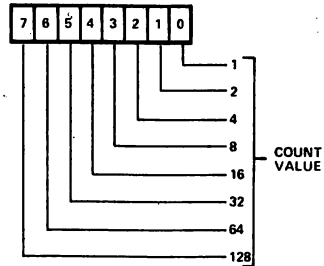
REG 12 – PERIPHERAL CONTROL REGISTER



Two bits are provided in the Auxiliary Control Register (bits 6 and 7) to allow selection of the T1 operating modes. The four possible modes are depicted in Figure H-17.

**Figure H-15: T1 Counter Registers**

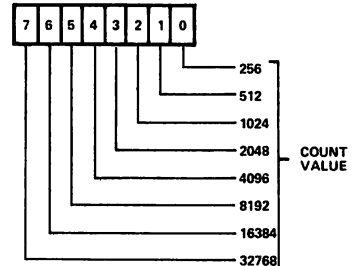
REG 4 – TIMER 1 LOW-ORDER COUNTER



**WRITE** – 8 BITS LOADED INTO T1 LOW-ORDER LATCHES. LATCH CONTENTS ARE TRANSFERRED INTO LOW-ORDER COUNTER AT THE TIME THE HIGH-ORDER COUNTER IS LOADED (REG 5).

**READ** – 8 BITS FROM T1 LOW-ORDER COUNTER TRANSFERRED TO MPU. IN ADDITION, T1 INTERRUPT FLAG IS RESET (BIT 6 IN INTERRUPT FLAG REGISTER).

REG 5 – TIMER 1 HIGH-ORDER COUNTER

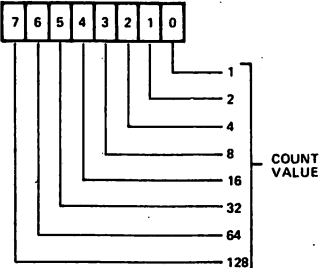


**WRITE** – 8 BITS LOADED INTO T1 HIGH-ORDER LATCHES. ALSO, AT THIS TIME BOTH HIGH AND LOW-ORDER LATCHES TRANSFERRED INTO T1 COUNTER. T1 INTERRUPT FLAG ALSO IS RESET.

**READ** – 8 BITS FROM T1 HIGH-ORDER COUNTER TRANSFERRED TO MPU.

**Figure H-16: T1 Latch Registers**

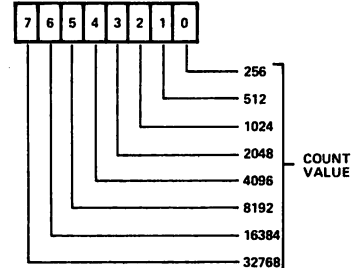
REG 6 – TIMER 1 LOW-ORDER LATCHES



**WRITE** – 8 BITS LOADED INTO T1 LOW-ORDER LATCHES. THIS OPERATION IS NO DIFFERENT THAN A WRITE INTO REG 4.

**READ** – 8 BITS FROM T1 LOW-ORDER LATCHES TRANSFERRED TO MPU. UNLIKE REG 4 OPERATION, THIS DOES NOT CAUSE RESET OF T1 INTERRUPT FLAG.

REG 7 – TIMER 1 HIGH-ORDER LATCHES

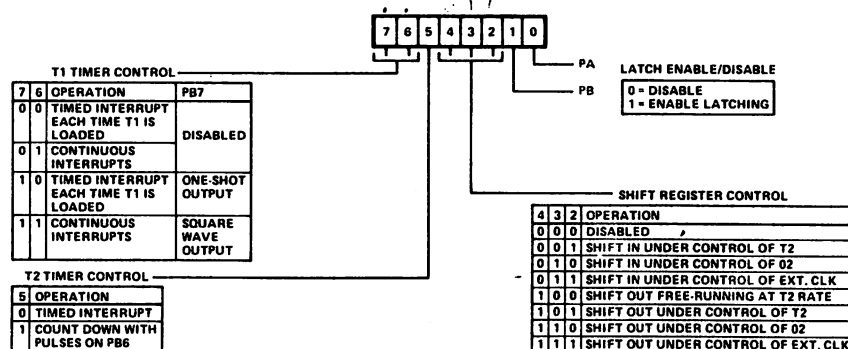


**WRITE** – 8 BITS LOADED INTO T1 HIGH-ORDER LATCHES. UNLIKE REG 4 OPERATION NO LATCH-TO-COUNTER TRANSFERS TAKE PLACE.

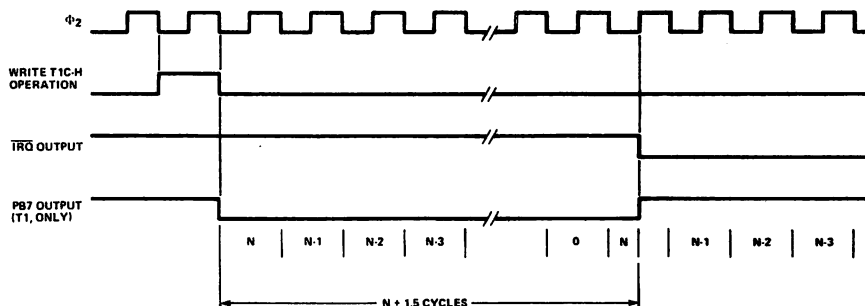
**READ** – 8 BITS FROM T1 HIGH-ORDER LATCHES TRANSFERRED TO MPU.

**Figure H-17: Auxiliary Control Register**

REG 11 – AUXILIARY CONTROL REGISTER



**Figure H-18: Timer 1 and Timer 2 One-Shot Mode Timing**



## TIMER 1 ONE-SHOT MODE

The interval timer one-shot mode allows generation of a single interrupt for each timer load operation. As with any interval timer, the delay between the “write T1C-H” operation and generation of the processor interrupt is a direct function of the data loaded into the timing counter. In addition to generating a single interrupt, Timer 1 can be programmed to produce a single negative pulse on the PB7 peripheral pin. With the output enabled (ACR7 = 1) a “write T1C-H” operation will cause PB7 to go low. PB7 will return high when Timer 1 times out. The result is a single programmable width pulse.

In the one-shot mode, writing into the high order latch has no effect on the operation of Timer 1. However, it will be necessary to assure that the low order latch contains the proper data before initiating the count-down with a “write T1C-H” operation. When the processor writes into the high order counter, the T1 Interrupt Flag will be cleared, the contents of the low order latch will be transferred into the low order counter, and the timer will begin to decrement at system clock rate. If the PB7 output is enabled, this signal will go low on the phase two following the write operation. When the counter reaches zero, the T1 Interrupt Flag will be set, the IRQ pin will go low

(interrupt enabled), and the signal on PB7 will go high. At this time the counter will continue to decrement at system clock rate. This allows the system processor to read the contents of the counter to determine the time since interrupt. However, the T1 Interrupt Flag cannot be set again unless it has been cleared as described in this specification.

Timing for the SY6522 interval timer one-shot modes is shown in Figure H-18.

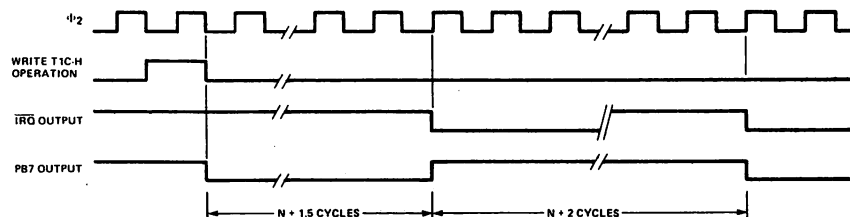
## TIMER 1 FREE-RUN MODE

The most important advantage associated with the latches in T1 is the ability to produce a continuous series of evenly spaced interrupts and the ability to produce a square wave on PB7 whose frequency is not affected by variations in the processor interrupt response time. This is accomplished in the "free-running" mode.

In the free-running mode, the Interrupt Flag is set and the signal on PB7 is inverted each time the counter reaches zero. However, instead of continuing to decrement from zero after a time-out, the timer automatically transfers the contents of the latch into the counter (16 bits) and continues to decrement from there. The Interrupt Flag can be cleared by writing T1C-H, by reading T1C-L, or by writing directly into the flag as described later. However, it is not necessary to rewrite the timer to enable setting the Interrupt Flag on the next time-out.

All interval timers in the SY6522 are "re-triggerable". Rewriting the counter will always re-initialize the time-out period. In fact, the time-out can be prevented completely if the processor continues to rewrite the timer before it reaches zero. Timer 1 will operate in this manner if the processor writes into the high order counter (T1C-H). However, by loading the latches only, the processor can access the timer during each down-counting operation without affecting the time-out in process. Instead, the data loaded into the latches will determine the length of the next time-out period. This capability is particularly valuable in the free-running mode with the output enabled. In this mode, the signal on PB7 is inverted and the interrupt flag is set with each time-out. By responding to the interrupts with new data for the latches, the processor can determine the period of the next half cycle during each half cycle of the output signal on PB7. In this manner, very complex waveforms can be generated. Timing for the free-running mode is shown in Figure H-19.

**Figure H-19: Timer 1 Free-Run Mode Timing**



Note: A precaution to take in the use of PB7 as the timer output concerns the Data Direction Register contents for PB7. Both DDRB bit 7 and ACR bit 7 must be 1 for PB7 to function as the timer output. If one is 1 and the other is 0, then PB7 functions as a normal output pin, controlled by ORB bit 7.

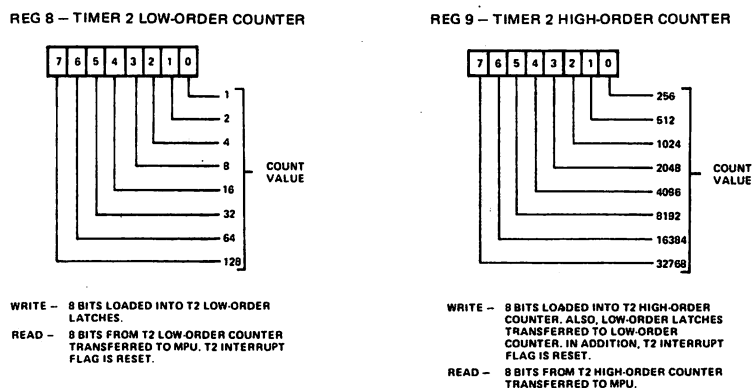
## TIMER 2 OPERATION

Timer 2 operates as an interval timer (in the "one-slot" mode only), or as a counter for counting negative pulses on the PB6 peripheral pin. A single control bit is provided in the Auxiliary Control Register to select between these two modes. This timer is comprised of a "write-only" low-order latch (T2L-L), a "read-only" low-order counter and a read/write high order counter. The counter registers act as a 16-bit counter which decrements at  $\phi 2$  rate. Figure H-20 illustrates the T2 Counter Registers.

## TIMER 2 ONE-SHOT MODE

As an interval timer, T2 operates in the "one-shot" mode similar to Timer 1. In this mode, T2 provides a single interrupt for each "write T2C-H" operation. After timing out, the counter will continue to decrement. However, setting of the Interrupt Flag will be disabled after initial time-out so that it will not be set by the counter continuing to decrement through zero. The processor must rewrite T2C-H to enable setting of the Interrupt Flag. The Interrupt Flag is cleared by reading T2C-L or by writing T2C-H. Timing for this operation is shown in Figure H-18.

**Figure H-20: T2 Counter Registers**



## TIMER 2 PULSE COUNTING MODE

In the pulse counting mode, T2 serves primarily to count a predetermined number of negative-going pulses on PB6. This is accomplished by first loading a number into T2. Writing into T2C-H clears the Interrupt Flag and allows the counter to decrement each time a pulse is applied to PB6. The Interrupt Flag will be set when T2 reaches zero. At this time the counter will continue to decrement with each pulse on PB6. However, it is necessary to rewrite T2C-H to allow the Interrupt Flag to set on subsequent down-counting operations. Timing for this mode is shown in Figure H-21. The pulse must be low on the leading edge of  $\phi 2$ .

## SHIFT REGISTER OPERATION

The Shift Register (SR) performs serial data transfers into and out of the CB2 pin under control of an internal modulo-8 counter. Shift pulses can be applied to the CB1 pin from an external source or, with the proper mode selection, shift pulses generated internally will appear on the CB1 pin for controlling external devices.

## INTERRUPT OPERATION

The control bits which select the various shift register operating modes are located in the Auxiliary Control Register. Figure H-22 illustrates the configuration of the SR data bits and the SR control bits of the ACR.

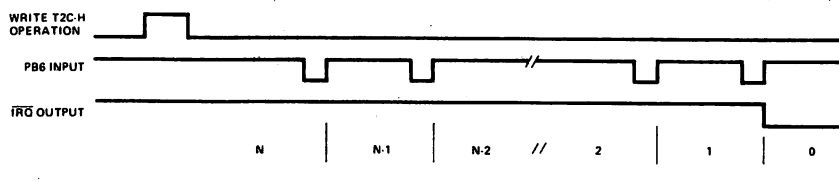
Figures H-23 and H-24 illustrate the operation of the various shift register modes.

Controlling interrupts within the SY6522 involves three principle operations. These are flagging the interrupts, enabling interrupts and signaling to the processor that an active interrupt exists within the chip. Interrupt flags are set by interrupting conditions which exist within the chip or on inputs to the chip. These flags normally remain set until the interrupt has been serviced. To determine the source of an interrupt, the microprocessor must examine these flags in order from highest to lowest priority. This is accomplished by reading the flag register into the processor accumulator, shifting this register either right or left and then using conditional branch instructions to detect an active interrupt.

Associated with each Interrupt Flag is an interrupt enable bit. This can be set or cleared by the processor to enable interrupting the processor from the corresponding Interrupt Flag. If an interrupt flag is set to a logic 1 by an interrupting condition, and the corresponding interrupt enable bit is set to a 1, the Interrupt Request Output (IRQ) will go low. IRQ is an "open-collector" output which can be "wire-or'ed" with other devices in the system to interrupt the processor.

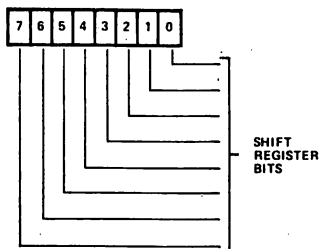
In the SY6522, all the Interrupt Flags are contained in one register. In addition, bit 7 of this register will be read as a logic 1 when an interrupt exists within the chip. This allows very convenient polling of several devices within a system to locate the source of an interrupt.

**Figure H-21: Timer 2 Pulse Counting Mode**



**Figure H-22: SR and ACR Control Bits**

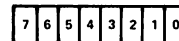
REG 10 — SHIFT REGISTER



**NOTES:**

1. WHEN SHIFTING OUT, BIT 7 IS THE FIRST BIT OUT AND SIMULTANEOUSLY IS ROTATED BACK INTO BIT 0.
2. WHEN SHIFTING IN, BITS INITIALLY ENTER BIT 0 AND ARE SHIFTED TOWARDS BIT 7.

REG 11 — AUXILIARY CONTROL REGISTER



SHIFT REGISTER  
MODE CONTROL

4	3	2	OPERATION
0	0	0	DISABLED
0	0	1	SHIFT IN UNDER CONTROL OF T2
0	1	0	SHIFT IN UNDER CONTROL OF $\phi_2$
0	1	1	SHIFT IN UNDER CONTROL OF EXT CLK
1	0	0	SHIFT OUT FREE-RUNNING AT T2 RATE
1	0	1	SHIFT OUT UNDER CONTROL OF T2
1	1	0	SHIFT OUT UNDER CONTROL OF $\phi_2$
1	1	1	SHIFT OUT UNDER CONTROL OF EXT CLK

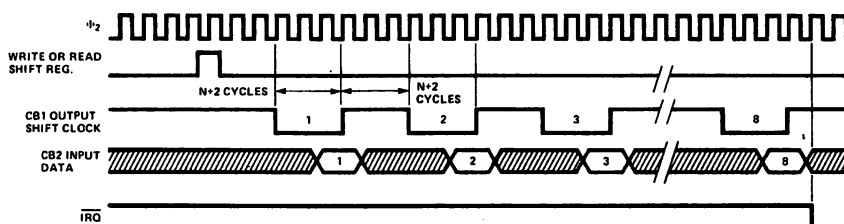
### SR Disabled (000)

The 000 mode is used to disable the Shift Register. In this mode the microprocessor can write or read the SR, but the shifting operation is disabled and operation of CB1 and CB2 is controlled by the appropriate bits in the Peripheral Control Register (PCR). In this mode the SR Interrupt Flag is disabled (held to a logic 0).

### Shift in Under Control of T2 (001)

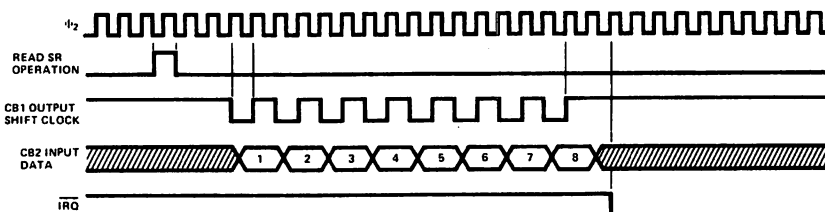
In the 001 mode the shifting rate is controlled by the low order 8 bits of T2. Shift pulses are generated on the CB1 pin to control shifting in external devices. The time between transitions of this output clock is a function of the system clock period and the contents of the low order T2 latch (N).

The shifting operation is triggered by writing or reading the shift register. Data is shifted first into the low order bit of SR and is then shifted into the next higher order bit of the shift register on the negative-going edge of each clock pulse. The input data should change before the positive-going edge of the CB1 clock pulse. This data is shifted into the shift register during the  $\phi_2$  clock cycle following the positive-going edge of the CB1 clock pulse. After 8 CB1 clock pulses, the shift register Interrupt Flag will be set and IRQ will go low.



### Shift in Under Control of $\phi_2$ (010)

In mode 010 the shift rate is a direct function of the system clock frequency. CB1 becomes an output which generates shift pulses for controlling external devices. Timer 2 operates as an independent interval timer and has no effect on SR. The shifting operation is triggered by reading or writing the Shift Register. Data is shifted first bit 0 and is then shifted into the next higher order bit of the shift register on the trailing edge of each  $\phi_2$  clock pulse. After 8 clock pulses, the shift register Interrupt Flag will be set, and the output clock pulses on CB1 will stop.



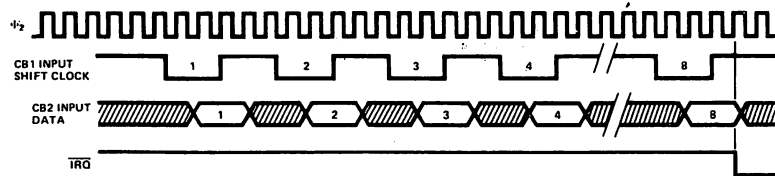
### Shift in Under Control of External CB1 Clock (011)

In mode 011 CB1 becomes an input. This allows an external device to load the shift register at its own pace. The shift register counter will interrupt the processor each time 8 bits have been shifted in. However, the shift register counter does not stop the shifting operation; it acts simply as a pulse counter. Reading or writing the Shift Register resets the Interrupt flag and initializes the SR counter to count another 8 pulses.



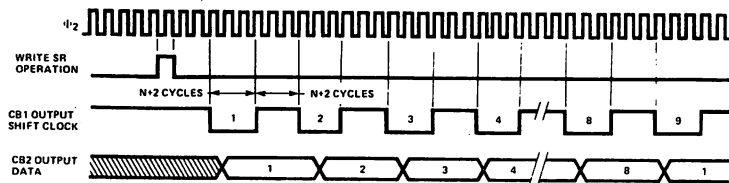
Note that the data is shifted during the first system clock cycle following the positive-going edge of the CB1 shift pulse. For this reason, data must be held stable during the first full cycle following CB1 going high.

**Figure H-23: Shift Register Input Modes**



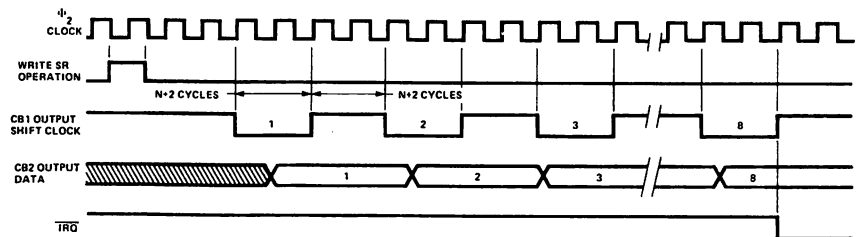
### Shift Out Free-Running at T2 Rate (100)

Mode 100 is very similar to mode 101 in which the shifting rate is set by T2. However, in mode 100 the SR counter does not stop the shifting operation. Since the Shift Register bit 7 (SR7) is recirculated back into bit 0, the 8 bits loaded into the shift register will be clocked onto CB2 repetitively. In this mode the shift register counter is disabled.



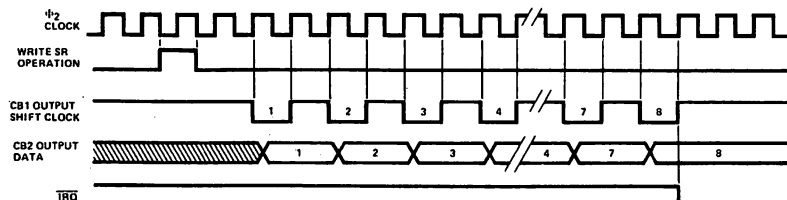
### Shift Out Under Control of T2 (101)

In mode 101 the shift rate is controlled by T2 (as in the previous mode). However, with each read or write of the shift register the SR Counter is reset and 8 bits are shifted onto CB2. At the same time, 8 shift pulses are generated on CB1 to control shifting in external devices. After the 8 shift pulses, the shifting is disabled, the SR Interrupt Flag is set and CB2 remains at the last data level.



### Shift Out Under Control of $\phi_2$ (110)

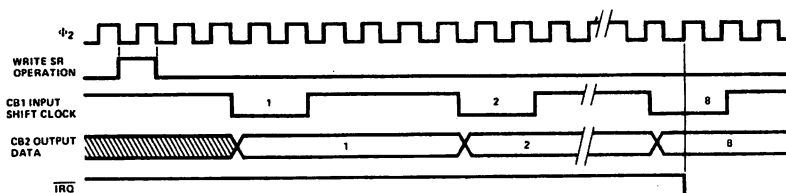
In mode 110, the shift rate is controlled by the  $\phi_2$  system clock.



## Shift Out Under Control of External CB1 Clock (111)

In mode 111 shifting is controlled by pulses applied to the CB1 pin by an external device. The SR counter sets the SR Interrupt flag each time it counts 8 pulses but it does not disable the shifting function. Each time the microprocessor writes or reads the shift register, the SR Interrupt flag is reset and the SR counter is initialized to begin counting the next 8 shift pulses on pin CB1. After 8 shift pulses, the Interrupt flag is set. The microprocessor can then load the shift register with the next byte of data.

**Figure H-24: Shift Register Output Modes**

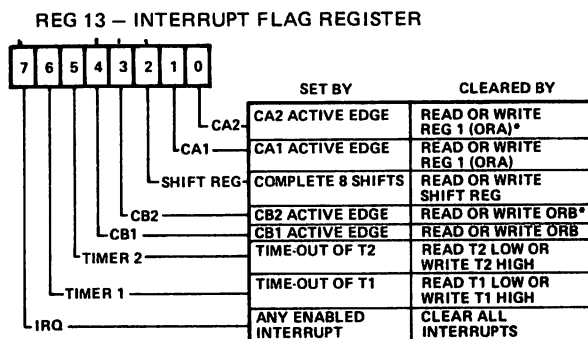


The Interrupt Flag Register (IFR) and Interrupt Enable Register (IER) are depicted in Figures H-25 and H-26, respectively.

The IFR may be read directly by the processor. In addition, individual flag bits may be cleared by writing a "1" into the appropriate bit of the IFR. When the proper chip select and register signals are applied to the chip, the contents of this register are placed on the data bus. Bit 7 indicates the status of the IRQ output. This bit corresponds to the logic function:  $IRQ = IFR6 \times IER6 + IFR5 \times IER5 + IFR4 \times IER4 + IFR3 \times IER3 + IFR2 \times IER2 + IFR1 \times IER1 + IFR0 \times IER0$ . Note: X = logic AND, + = Logic OR.

The IFR bit 7 is not a flag. Therefore, this bit is not directly cleared by writing a logic 1 into it. It can only be cleared by clearing all the flags in the register or by disabling all the active interrupts as discussed in the next section.

**Figure H-25: Interrupt Flag Register (IFR)**



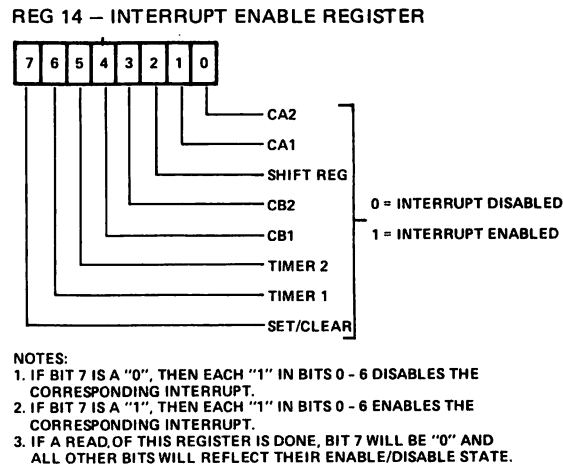
\* IF THE CA2/CB2 CONTROL IN THE PCR IS SELECTED AS "INDEPENDENT" INTERRUPT INPUT, THEN READING OR WRITING THE OUTPUT REGISTER ORA/ORB WILL NOT CLEAR THE FLAG BIT. INSTEAD, THE BIT MUST BE CLEARED BY WRITING INTO THE IFR, AS DESCRIBED PREVIOUSLY.

For each Interrupt Flag in IFR, there is a corresponding bit in the Interrupt Enable Register. The system processor can be set or clear selected bits in this register to facilitate controlling individual interrupts without affecting others. This is accomplished by writing to address 1110 (IER address). If bit 7 of the data placed on the system data bus during this write operation is a 0, each 1 in bits 6 through 0 clears the corresponding bit in the Interrupt Enable Register. For each zero in bits 6 through 0, the corresponding bit is unaffected.

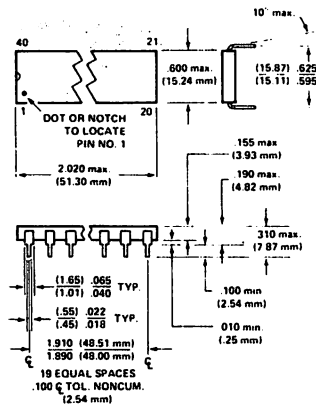
Setting selected bits in the Interrupt Enable Register is accomplished by writing to the same address with bit 7 in the data word set to a logic 1. In this case, each 1 in bits 6 through 0 will set the corresponding bit. For each zero, the corresponding bit will be unaffected. This individual control of the setting and clearing operations allows very convenient control of the interrupts during system operation.

In addition to setting and clearing IER bits, the processor can read the contents of this register by placing the proper address on the register select and chip select inputs with the R/W line high. Bit 7 will be read as a logic 0.

**Figure H-26: Interrupt Enable Register (IER)**



# **PACKAGE OUTLINE**



NOTE: Pin No. 1 is in lower left corner when symbolization is in normal orientation

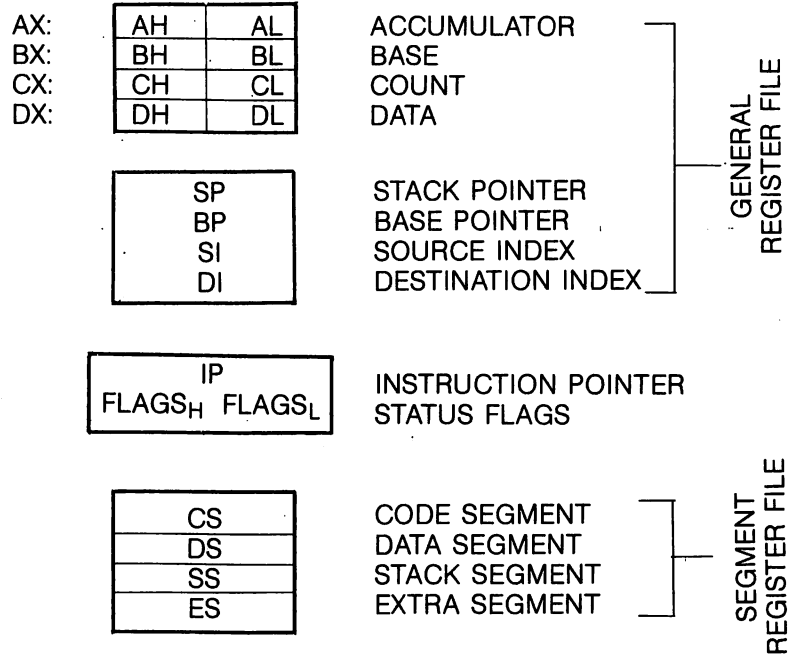
# **ORDERING INFORMATION**

Order Number	Package Type	Frequency Option
SYP 6522	Plastic	1 MHz
SYP 6522A	Plastic	2 MHz
SYC 6522	Ceramic	1 MHz
SYC 6522A	Ceramic	2 MHz

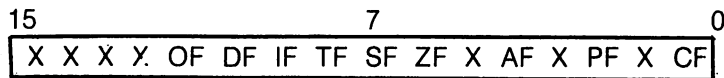
# **PIN CONFIGURATION**

VSS	1	40	CA1
PA0	2	39	CA2
PA1	3	38	RS0
PA2	4	37	RS1
PA3	5	36	RS2
PA4	6	35	RS3
PA5	7	34	RES
PA6	8	33	D0
PA7	9	32	D1
PB0	10	SV6522 31	D2
PB1	11	30	D3
PB2	12	29	D4
PB3	13	28	D5
PB4	14	27	D6
PB5	15	26	D7
PB6	16	25	+2
PB7	17	24	CS1
CB1	18	23	CS2
CB2	19	22	R/W
VCC	20	21	IRG

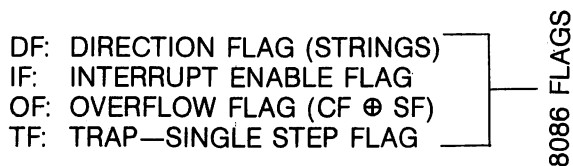
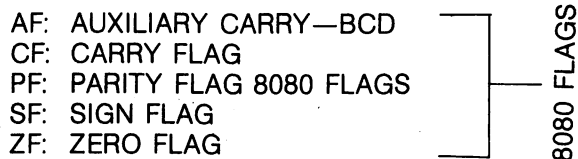
**I.1 8086 REGISTER MODEL**



Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:



X = Don't care



All mnemonics ©Intel Corporation 1981.

## I.2 OPERAND SUMMARY

"REG" FIELD BIT ASSIGNMENTS

16-BIT(W=1)	8-BIT(W=0)	SEGMENT
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

## I.3 SECOND INSTRUCTION BYTE SUMMARY

mod xxx r/m

MOD	DISPLACEMENT
00	DISP=0*; disp-low and disp-high are absent
01	DISP=disp-low sign-extended to 16-bits, disp-high is absent
10	DISP=disp-high:disp-low
11	r/m is treated as a "reg" field

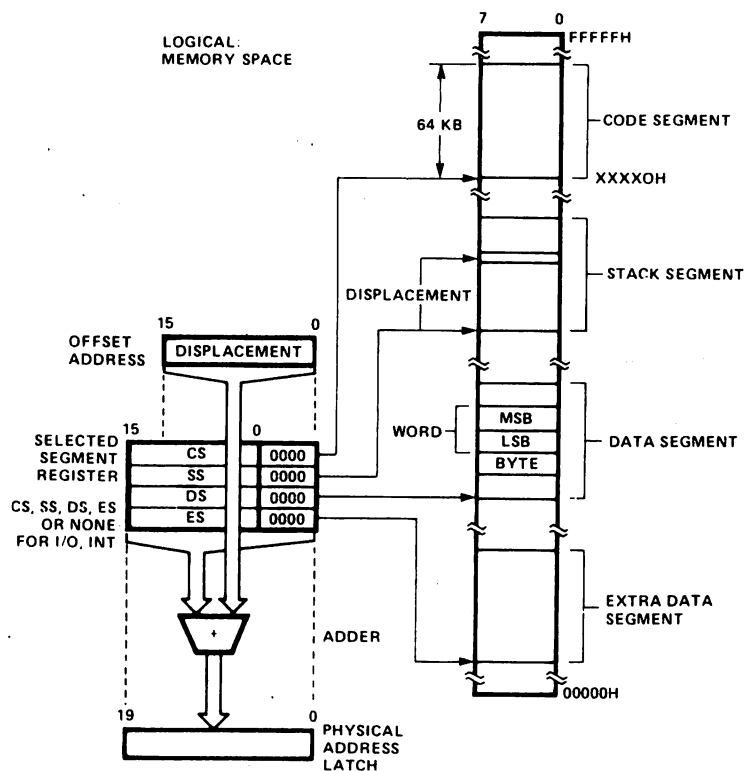
R/M	OPERAND ADDRESS	DEFAULT SEGMENT
000	(BX) + (SI) + DISP	DS
001	(BX) + (DI) + DISP	DS
010	(BP) + (SI) + DISP	SS
011	(BP) + (DI) + DISP	SS
100	(SI) + DISP	DS
101	(DI) + DISP	DS
110	(BP) + DISP*	SS
111	(BX) + DISP	DS

DISP follows 2nd byte of instruction (before data if required).  
 \*except if mod=00 and r/m=110; then EA=disp-high: disp-low.

OPERAND ADDRESS (EA) TIMING (CLOCKS):  
 Add 4 clocks for word operands at ODD ADDRESSES.  
 Immed offset=6  
 Base (BX, BP, SI, DI)=5  
 Base + DISP=9  
 Base + index (BP + DI, BX + SI)=7  
 Base + index (BP + SI, BX + DI)=8  
 Base + index (BP + DI, BX + SI) + DISP=11  
 Base + index (BP + SI, BX + DI) + DISP=12

All mnemonics ©Intel Corporation 1981.

## I.4 MEMORY SEGMENTATION MODEL



### SEGMENT OVERRIDE PREFIX

0 0 1 REG 1 1 0

Timing: 2 clocks

### USE OF SEGMENT OVERRIDE

OPERAND REGISTER	DEFAULT	WITH OVERRIDE PREFIX
IP (code address)	CS	Never
SP (stack address)	SS	Never
BP (stack address or stack marker)	SS	BP + DS, or ES, or CS
SI or DI (not incl. strings)	DS	ES, SS, or CS
SI (implicit source addr. for strings)	DS	ES, SS, or CS
DI (implicit dest. addr. for strings)	ES	Never

## I.5 INSTRUCTION SET DATA

Section I.5.2 presents instruction set data, grouped by function. Section I.9 provides an alphabetic index to the data.

All mnemonics ©Intel Corporation 1981.

## I.5.1 KEY TO FLAG EFFECTS

The following key refers to the flag sections in the instruction set data in Section I.5.2.

### FLAG EFFECT KEY

IDENTIFIER	EXPLANATION
(blank)	Not altered
0	Cleared to 0
1	Set to 1
X	Set or cleared according to result
U	Undefined—contains no reliable value
R	Restored from previously-saved value

## I.5.2 DATA TRANSFER

### MOV=Move

Flags: O D I T S Z A P C

Register/memory to/from register

1 0 0 0 1 0 d w	mod reg r/m
-----------------	-------------

Timing (clocks): register to register 2  
memory to register 8+EA  
register to memory 9+EA

Immediate to register/memory

1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w=1
-----------------	---------------	------	-------------

Timing: 10+EA clocks

Immediate to register

1 0 1 1 w reg	data	data if w=1
---------------	------	-------------

Timing: 4 clocks

Memory to accumulator

1 0 1 0 0 0 0 w	addr-low	addr-high
-----------------	----------	-----------

Timing: 10 clocks

Accumulator to memory

1 0 1 0 0 0 1 w	addr-low	addr-high
-----------------	----------	-----------

Timing: 10 clocks

Register/memory to segment register

1 0 0 0 1 1 1 0	mod 0 reg r/m
-----------------	---------------

Timing (clocks): register to register 2  
memory to register 8+EA

Segment register to register/memory

1 0 0 0 1 1 0 0	mod 0 reg r/m
-----------------	---------------

All mnemonics ©Intel Corporation 1981.



**PUSH=Push**

Timing (clocks): register to register 2  
register to memory 9+EA

Flags: O D I T S Z A P C

Register/memory

1 1 1 1 1 1 1 1	mod 1 1 0 r/m
-----------------	---------------

Timing (clocks): register 10  
memory 16+EA

0 1 0 1 0 reg
---------------

Timing: 10 clocks

Segment register

0 0 0 reg 1 1 0
-----------------

Timing: 10 clocks

**POP=Pop**

Flags: O D I T S Z A P C

Register/memory

1 0 0 0 1 1 1 1	mod 0 0 0 r/m
-----------------	---------------

Timing (clocks): register 8  
memory 17+EA

Register

0 1 0 1 1 reg
---------------

Timing: 8 clocks

Segment register

0 0 0 reg 1 1 1
-----------------

Timing: 8 clocks

**XCHG=Exchange**

Flags: O D I T S Z A P C

Register/memory with register

1 0 0 0 0 1 1 w	mod reg r/m
-----------------	-------------

Timing (clocks): register with register 4  
memory with register 17+EA

Register with accumulator

1 0 0 1 0 reg
---------------

Timing: 3 clocks

**IN=Input to AL/AX from**

Flags: O D I T S Z A P C

Fixed Port

1 1 1 0 0 1 0 w	port
-----------------	------

Timing: 10 clocks

Variable port (DX)

1 1 1 0 1 1 0 w

Timing: 8 clocks

**OUT=Output from  
AL/AX to**

Flags: O D I T S Z A P C

Fixed Port

1 1 1 0 0 1 1 w port

Timing: 10 clocks

Variable port (DX)

1 1 1 0 1 1 1 w

Timing: 8 clocks

**XLAT=Translate Byte  
to AL**

Flags: O D I T S Z A P C

1 1 0 1 0 1 1 1

Timing: 11 clocks

**LEA=Load EA to  
Register**

Flags: O D I T S Z A P C

1 0 0 0 1 1 0 1 mod reg r/m

Timing: 2+EA clocks

**LDS=Load Pointer to  
DS**

Flags: O D I T S Z A P C

1 1 0 0 0 1 0 1 mod reg r/m

Timing: 16+EA clocks

**LES=Load Pointer to  
ES**

Flags: O D I T S Z A P C

1 1 0 0 0 1 0 0 mod reg r/m

Timing: 16+EA clocks

**LAHF=Load AH with  
Flags**

Flags: O D I T S Z A P C

1 0 0 1 1 1 1 1

Timing: 4 clocks

**SAHF=Store AH into  
Flags**

Flags: O D I T S Z A P C  
R R R R R

1 0 0 1 1 1 1 0

Timing: 4 clocks

All mnemonics ©Intel Corporation 1981.

**PUSHF=Push Flags**

Flags: O D I T S Z A P C

1 0 0 1 1 1 0 0

Timing: 10 clocks

**POPF=Pop Flags**Flags: O D I T S Z A P C  
R R R R R R R R R

1 0 0 1 1 1 0 1

Timing: 8 clocks

**I.5.3 ARITHMETIC****ADD=Add**Flags: O D I T S Z A P C  
X X X X X X X X

Reg./memory with register to either

0 0 0 0 0 d w mod reg r/m

Timing (clocks): register to register 3  
memory to register 9+EA  
register to memory 16+EA

Immediate to register/memory

1 0 0 0 0 s w mod 0 0 0 r/m data data if s:w=01

Timing (clocks): immediate to register 4  
immediate to memory 17+EA

Immediate to accumulator

0 0 0 0 0 1 0 w data data if w=1

Timing: 4 clocks

**ADC=Add with Carry**Flags: O D I T S Z A P C  
X X X X X X X X

Reg./memory with register to either

0 0 0 1 0 d w mod reg r/m

Timing (clocks): register to register 3  
memory to register 9+EA  
register to memory 16+EA

Immediate to register/memory

1 0 0 0 0 s w mod 0 1 0 r/m data data if s:w=01

Timing (clocks): immediate to register 4  
immediate to memory 17+EA

All mnemonics ©Intel Corporation 1981.

Immediate to accumulator

0 0 0 1 0 1 0 w	data	data if w=1
-----------------	------	-------------

Timing: 4 clocks

**INC=Increment**

Flags: O D I T S Z A P C  
X X X X X X X

Register/memory

1 1 1 1 1 1 1 w	mod 0 0 0 r/m
-----------------	---------------

Timing (clocks): register 2  
memory 15+EA

Register

0 1 0 0 0 reg
---------------

Timing: 2 clocks

**AAA=ASCII Adjust for Add**

Flags: O D I T S Z A P C  
U U X U X

0 0 1 1 0 1 1 1
-----------------

Timing: 4 clocks

**DAA=Decimal Adjust for Add**

Flags: O D I T S Z A P C  
X X X X X X

0 0 1 0 0 1 1 1
-----------------

Timing: 4 clocks

**SUB=Subtract**

Flags: O D I T S Z A P C  
X X X X X X

0 0 1 0 1 0 d w	mod reg r/m
-----------------	-------------

Timing (clocks): register from register 3  
memory from register 9+EA  
register from memory 16+EZ

Immediate from register/memory

1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s:w=01
-----------------	---------------	------	----------------

Timing (clocks): immediate from register 4  
immediate from memory 17+EA

Immediate from accumulator

0 0 1 0 1 1 0 w	data	data if w=1
-----------------	------	-------------

Timing: 4 clocks

All mnemonics ©Intel Corporation 1981.

**SBB=Subtract with Borrow**

Flags: O D I T S Z A P C  
X X X X X X X

0 0 0 1 1 0 d w	mod reg r/m
-----------------	-------------

Timing (clocks): register from register 3  
memory from register 9+EA  
register from memory 16+EA

Immediate from register/memory

1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s:w=01
-----------------	---------------	------	----------------

Timing (clocks): immediate from register 4  
immediate from memory 17+EA

Immediate from accumulator

0 0 0 1 1 1 0 w	data	data if w=1
-----------------	------	-------------

Timing: 4 clocks

**DEC=Decrement**

Flags: O D I T S Z A P C  
X X X X X X

Register/memory

1 1 1 1 1 1 1 w	mod 0 0 1 r/m
-----------------	---------------

Timing (clocks): register 2  
memory 15+EA

Register

0 1 0 0 1 reg
---------------

Timing: 2 clocks

**NEG=Change Sign**

Flags: O D I T S Z A P C  
X X X X X 1\*

\*0 if destination=0

1 1 1 1 0 1 1 w	mod 0 1 1 r/m
-----------------	---------------

Timing (clocks): register 3  
memory 16+EA

**CMP=Compare**

Flags: O D I T S Z A P C  
X X X X X X

Register/memory and register

0 0 1 1 1 0 d w	mod reg r/m
-----------------	-------------

Timing (clocks): register with register 3  
memory with register 9+EA  
register with memory 9+EA

All mnemonics ©Intel Corporation 1981.

### Immediate with register/memory

1 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s:w=01
---------------	---------------	------	----------------

Timing (clocks): immediate with register 4  
immediate with memory 17+EA

### Immediate with accumulator

0 0 1 1 1 0 w	data	data if w=1
---------------	------	-------------

Timing: 4 clocks

### AAS=ASCII Adjust for Subtract

Flags: O D I T S Z A P C  
U U X U X

0 0 1 1 1 1 1
---------------

Timing: 4 clocks

### DAS=Decimal Adjust for Subtract

Flags: O D I T S Z A P C  
U X X X X X

0 0 1 0 1 1 1
---------------

Timing: 4 clocks

### MUL=Multiply (Unsigned)

Flags: O D I T S Z A P C  
X U U U U X

1 1 1 0 1 1 w	mod 1 0 0 r/m
---------------	---------------

Timing (clocks): 8-bit 71+EA  
16-bit 124+EA

### IMUL=Integer Multiply (Signed)

Flags: O D I T S Z A P C  
X U U U U X

1 1 1 0 1 1 w	mod 1 0 1 r/m
---------------	---------------

Timing (clocks): 8-bit 90+EA  
16-bit 144+EA

### AAM=ASCII Adjust for Multiply

Flags: O D I T S Z A P C  
U X X U X U

1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0
-----------------	-----------------

Timing: 83 clocks

### DIV=Divide (Unsigned)

Flags: O D I T S Z A P C  
U U U U U U

1 1 1 0 1 1 w	mod 1 1 0 r/m
---------------	---------------

Timing (clocks): 8-bit 90+EA  
16-bit 155+EA

All mnemonics ©Intel Corporation 1981.

**IDIV=Integer Divide  
(Signed)**

Flags: O D I T S Z A P C  
U U U U U U U

1 1 1 1 0 1 1 w mod 1 1 1 r/m

Timing (clocks): 8-bit 112+EA  
16-bit 177+EA

**AAD=ASCII Adjust for  
Divide**

Flags: O D I T S Z A P C  
U X X U X U

1 1 0 1 0 1 0 1 0 0 0 0 1 0 1 0

Timing: 60 clocks

**CBW=Convert Byte to  
Word**

Flags: O D I T S Z A P C

1 0 0 1 1 0 0 0

Timing: 2 clocks

**CWD=Convert Word  
to Double Word**

Flags: O D I T S Z A P C

1 0 0 1 1 0 0 1

Timing: 5 clocks

## I.5.4 LOGIC

**NOT=Invert**

Flags: O D I T S Z A P C

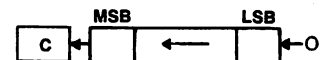
1 1 1 1 0 1 1 w mod 0 1 0 r/m

Timing (clocks): register 3  
memory 16+EA

**SHL/SAL=Shift  
Logical/Arithmetic  
Left**

Flags: O D I T S Z A P C  
X X

1 1 0 1 0 0 v w mod 1 0 0 r/m

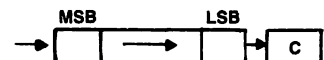


Timing (clocks): single-bit register 2  
single-bit memory 15+EA  
variable-bit register 8+4/bit  
variable-bit memory 20+EA+4/bit

**SHR=Shift Logical  
Right**

Flags: O D I T S Z A P C  
X X

1 1 0 1 0 0 v w mod 1 0 1 r/m



Timing (clocks): single-bit register 2  
single-bit memory 15+EA  
variable-bit register 8+4/bit  
variable-bit memory 20+EA+4/bit

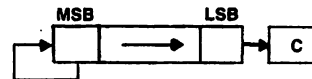
All mnemonics ©Intel Corporation 1981.

**SAR=Shift Arithmetic Right**

Flags: O D I T S Z A P C  
X X X U X X

1 1 0 1 0 0 v w mod 1 1 1 r/m

Timing (clocks): single-bit register 2  
single-bit memory 15+EA  
variable-bit register 8+4/bit  
variable-bit memory 20+EA+4/bit

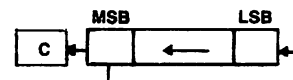


**ROL=Rotate Left**

Flags: O D I T S Z A P C  
X X X X X

1 1 0 1 0 0 v w mod 0 0 0 r/m

Timing (clocks): single-bit register 2  
single-bit memory 15+EA  
variable-bit register 8+4/bit  
variable-bit memory 20+EA+4/bit

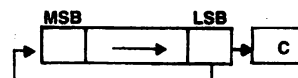


**ROR=Rotate Right**

Flags: O D I T S Z A P C  
X X X X

1 1 0 1 0 0 v w mod 0 0 1 r/m

Timing (clocks): single-bit register 2  
single-bit memory 15+EA  
variable-bit register 8+4/bit  
variable-bit memory 20+EA+4/bit

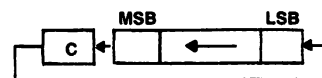


**RCL=Rotate Through Carry Left**

Flags: O D I T S Z A P C  
X X X X

1 1 0 1 0 0 v w mod 0 1 0 r/m

Timing (clocks): single-bit register 2  
single-bit memory 15+EA  
variable-bit register 8+4/bit  
variable-bit memory 20+EA+4/bit

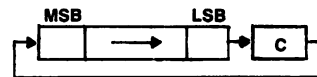


**RCR=Rotate Through Carry Right**

Flags: O D I T S Z A P C  
X X X X

1 1 0 1 0 0 v w mod 0 1 1 r/m

Timing (clocks): single-bit register 2  
single-bit memory 15+EA  
variable-bit register 8+4/bit  
variable-bit memory 20+EA+4/bit





**AND=And**

Flags: O D I T S Z A P C  
 O X X U X O

Reg./memory and register to either

0 0 1 0 0 0 d w	mod reg r/m
-----------------	-------------

Timing (clocks): register to register 3  
 memory to register 9+EA  
 register to memory 16+EA

Immediate to register/memory

1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w=1
-----------------	---------------	------	-------------

Timing (clocks): immediate to register 4  
 immediate to memory 17+EA

Immediate to accumulator

0 0 1 0 0 1 0 w	data	data if w=1
-----------------	------	-------------

Timing: 4 clocks

Flags: O D I T S Z A P C  
 O X X U X O

**TEST=And Function to Flags, No Result**

Register/memory and register

1 0 0 0 0 1 0 w	mod reg r/m
-----------------	-------------

Timing (clocks): register to register 3  
 register with memory 9+EA

Immediate data and register/memory

1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w=1
-----------------	---------------	------	-------------

Timing (clocks): immediate with register 4  
 immediate with memory 10+EA

Immediate data and accumulator

1 0 1 0 1 0 0 w	data	data if w=1
-----------------	------	-------------

Timing: 4 clocks

**OR=Or**

Flags: O D I T S Z A P C  
 O X X U X O

Reg./memory and register to either

0 0 0 0 1 0 d w	mod reg r/m
-----------------	-------------

Timing (clocks): register to register 3  
 memory to register 9+EA  
 register to memory 16+EA

All mnemonics ©Intel Corporation 1981.

Immediate to register/memory

1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w=1
-----------------	---------------	------	-------------

Timing (clocks): immediate to register 4  
immediate to memory 17+EA

Immediate to accumulator

0 0 0 0 1 1 0 w	data	data if w=1
-----------------	------	-------------

Timing: 4 clocks

**XOR=Exclusive Or**

Flags: O D I T S Z A P C  
O X X U X O

Reg./memory and register to either

0 0 1 1 0 0 d w	mod reg r/m
-----------------	-------------

Timing (clocks): register to register 3  
memory to register 9+EA  
register to memory 16+EA

Immediate to register/memory

1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w=1
-----------------	---------------	------	-------------

Timing (clocks): immediate to register 4  
immediate to memory 17+EA

Immediate to accumulator

0 0 1 1 0 1 0 w	data	data if w=1
-----------------	------	-------------

Timing: 4 clocks

## I.5.5 STRING MANIPULATION

**REP=Repeat**

Flags: O D I T S Z A P C

1 1 1 1 0 0 1 z
-----------------

Timing: 6 clocks/loop

**MOVS=Move String**

Flags: O D I T S Z A P C

1 0 1 0 0 1 0 w
-----------------

Timing: 17 clocks

**CMPS=Compare String**

Flags: O D I T S Z A P C  
X X X X X

1 0 1 0 0 1 1 w
-----------------

Timing: 22 clocks

All mnemonics ©Intel Corporation 1981.

**SCAS=Scan String**

Flags: O D I T S Z A P C  
 X X X X X X

1 0 1 0 1 1 1 w
-----------------

Timing: 15 clocks

**LODS=Load String**

Flags: O D I T S Z A P C

1 0 1 0 1 1 0 w
-----------------

Timing: 12 clocks

**STOS=Store String**

Flags: O D I T S Z A P C

1 0 1 0 1 0 1 w
-----------------

Timing: 10 clocks

---

**I.5.6 CONTROL TRANSFER**

NOTE: Queue reinitialization is not included in the timing information for transfer operations. To account for instruction loading, add 8 clocks to timing numbers.

**CALL=Call**

Flags: O D I T S Z A P C

Direct within segment

1 1 1 0 1 0 0 0	disp-low	disp-high
-----------------	----------	-----------

Timing: 11 clocks

Indirect within segment

1 1 1 1 1 1 1 1	mod 0 1 0 r/m
-----------------	---------------

Timing: 13+EA clocks

Direct intersegment

1 0 0 1 1 0 1 0	offset-low	offset-high
	seg-low	seg-high

Timing: 20 clocks

Indirect intersegment

1 1 1 1 1 1 1 1	mod 0 1 1 r/m
-----------------	---------------

Timing: 29+EA clocks

**JMP=Unconditional Jump**

Flags: O D I T S Z A P C

Direct within segment

1 1 1 0 1 0 0 1	disp-low	disp-high
-----------------	----------	-----------

Timing: 7 clocks

---

All mnemonics ©Intel Corporation 1981.

Direct within segment-short

1 1 1 0 1 0 1 1	disp
-----------------	------

Timing: 7 clocks

Indirect within segment

1 1 1 1 1 1 1 1	mod 1 0 0	r/m
-----------------	-----------	-----

Timing: 7+EA clocks

Direct intersegment

1 1 1 0 1 0 1 0	offset-low	offset-high
	seg-low	seg-high

Timing: 7 clocks

Indirect intersegment

1 1 1 1 1 1 1 1	mod 1 0 1	r/m
-----------------	-----------	-----

Timing: 16+EA clocks

Flags: O D I T S Z A P C

**RET=Return from  
CALL**

Within segment

1 1 0 0 0 0 1 1
-----------------

Timing: 8 clocks

Within seg. adding immediate to SP

1 1 0 0 0 0 1 0	data-low	data-high
-----------------	----------	-----------

Timing: 12 clocks

Intersegment

1 1 0 0 1 0 1 1
-----------------

Timing: 18 clocks

Intersegment, adding immediate to SP

1 1 0 0 1 0 1 0	data-low	data-high
-----------------	----------	-----------

Timing: 17 clocks

**JE/JZ=Jump on  
Equal/Zero**

Flags: O D I T S Z A P C

0 1 1 1 0 1 0 0	disp
-----------------	------

Timing (clocks): jump is taken  
jump is not taken

8  
4

All mnemonics ©Intel Corporation 1981.

**JL/JNGE=Jump on Less/Not Greater or Equal**

Flags: O D I T S Z A P C

0 1 1 1 1 1 0 0	disp
-----------------	------

Timing (clocks): jump is taken	8
jump is not taken	4

**JLE/JNG=Jump on Less or Equal/Not Greater**

Flags: O D I T S Z A P C

0 1 1 1 1 1 1 0	disp
-----------------	------

Timing (clocks): jump is taken	8
jump is not taken	4

**JB/JNAE=Jump on Below/Not Above or Equal**

Flags: O D I T S Z A P C

0 1 1 1 0 0 1 0	disp
-----------------	------

Timing (clocks): jump is taken	8
jump is not taken	4

**JBE/JNA=Jump on Below or Equal/Not Above**

Flags: O D I T S Z A P C

0 1 1 1 0 1 1 0	disp
-----------------	------

Timing (clocks): jump is taken	8
jump is not taken	4

**JP/JPE=Jump on Parity/Parity Even**

Flags: O D I T S Z A P C

0 1 1 1 1 0 1 0	disp
-----------------	------

Timing (clocks): jump is taken	8
jump is not taken	4

**JO=Jump on Overflow**

Flags: O D I T S Z A P C

0 1 1 1 0 0 0 0	disp
-----------------	------

Timing (clocks): jump is taken	8
jump is not taken	4

**JS=Jump on Sign**

Flags: O D I T S Z A P C

0 1 1 1 1 0 0 0	disp
-----------------	------

Timing (clocks): jump is taken	8
jump is not taken	4

**JNE/JNZ=Jump on Not Equal/Not Zero**

Flags: O D I T S Z A P C

0 1 1 1 0 1 0 1	disp
-----------------	------

Timing (clocks): jump is taken	8
jump is not taken	4

-----  
All mnemonics ©Intel Corporation 1981.

Flags: O D I T S Z A P C

0 1 1 1 1 0 1	disp
---------------	------

Timing (clocks):	jump is taken	8
	jump is not taken	4

**JNLE/JG=Jump on  
Not Less or  
Equal/Greater**

Flags: O D I T S Z A P C

0 1 1 1 1 1 1 1	disp
-----------------	------

Timing (clocks):	jump is taken	8
	jump is not taken	4

**JNB/JAE=Jump on  
Not Below/Above or  
Equal**

Flags: O D I T S Z A P C

0 1 1 1 0 0 1 1	disp
-----------------	------

Timing (clocks):	jump is taken	8
	jump is not taken	4

**JNBE/JA=Jump on  
Not Below or  
Equal/Above**

Flags: O D I T S Z A P C

0 1 1 1 0 1 1 1	disp
-----------------	------

Timing (clocks):	jump is taken	8
	jump is not taken	4

**JNP/JPO=Jump on  
Not Parity/Parity Odd**

Flags: O D I T S Z A P C

0 1 1 1 1 0 1 1	disp
-----------------	------

Timing (clocks):	jump is taken	8
	jump is not taken	4

**JNO=Jump on Not Overflow**

Flags: O D I T S Z A P C

0 1 1 1 0 0 0 1	disp
-----------------	------

Timing (clocks): jump is taken	8
jump is not take	4

**JNS=Jump on Not Sign**

Flags: O D I T S Z A P C

0 1 1 1 1 0 0 1	disp
-----------------	------

Timing (clocks):	jump is taken	8
	jump is not taken	4

**LOOP=Loop CX  
Times**

Flags: O D I T S Z A P C

1 1 1 0 0 0 1 0	disp
-----------------	------

Timing (clocks):	jump is taken	9
	jump is not taken	5

All mnemonics © Intel Corporation 1981.

**LOOPZ/LOOPE=Loop  
While Zero/Equal**

Flags: O D I T S Z A P C

1 1 1 0 0 0 1	disp
---------------	------

Timing (clocks): jump is taken 11  
jump is not taken 5

**LOOPNZ/LOOPNE=Loop  
While Not Zero/Not  
Equal**

Flags: O D I T S Z A P C

1 1 1 0 0 0 0	disp
---------------	------

Timing (clocks): jump is taken 11  
jump is not taken 5

**JCXZ=Jump on CX  
Zero**

Flags: O D I T S Z A P C

1 1 1 0 0 0 1 1	disp
-----------------	------

Timing (clocks): jump is taken 9  
jump is not taken 5

**8086 Conditional Transfer Operations**

INSTRUCTION	CONDITION	INTERPRETATION
JE or JZ	ZF=1	"equal" or "zero"
JL or JNGE	(SR xor OF)=1	"less" or "not greater or equal"
JLE or JNG	((SP xor OF) or ZF)=1	"less or equal" or "not greater"
JB or JNAE	CF=1	"below" or "not above or equal"
JBE or JNA	(CF or ZF)=1	"below or equal" or "not above"
JP or JPE	PF=1	"parity" or "parity even"
JO	OF=1	"overflow"
JS	SF=1	"sign"
JNE or JNZ	ZF=0	"not equal" or "not zero"
JNL or JGE	(SF xor OF)=0	"not less" or "greater or equal"
JNLE or JG	((SF xor OF) or ZF)=0	"not less or equal" or "greater"
JNB or JAE	CF=0	"not below" or "above or equal"
JNBE or JA	(CF or ZF)=0	"not below or equal" or "above"
JNP or JPO	PF=0	"not parity" or "parity odd"
JNO	OF=0	"not overflow"
JNS	OF=0	"not sign"

NOTE: "Above and below" refer to the relation between two unsigned values, while "greater" and "less" refer to the relation between two signed values.

**INT=Interrupt**

Flags: O D I T S Z A P C  
O O

Type specified

1 1 0 0 1 1 0 1	type
-----------------	------

Timing: 50 clocks

All mnemonics © Intel Corporation 1981.

Type 3

11001100

Timing: 51 clocks

**INTO=Interrupt on  
Overflow**

Flags: O D I T S Z A P C  
O O

11001110

Timing: 52 clocks if pass 4 clocks if fail

**IRET=Interrupt Return**

Flags: O D I T S Z A P C  
R R R R R R R R R

11001111

Timing: 24 clocks

### I.5.7 PROCESSOR CONTROL

**CLC=Clear Carry**

Flags: O D I T S Z A P C  
O

11111000

Timing: 2 clocks

**STC=Set Carry**

Flags: O D I T S Z A P C  
1

11111001

Timing: 2 clocks

**CMC=Complement  
Carry**

Flags: O D I T S Z A P C  
X

11110101

Timing: 2 clocks

**NOP=No Operation**

Flags: O D I T S Z A P C

10010000

Timing: 3 clocks

**CLD=Clear Direction**

Flags: O D I T S Z A P C  
O

11111100

Timing: 2 clocks

All mnemonics ©Intel Corporation 1981.



**STD=Set Direction**

Flags: O D I T S Z A P C  
          1

1 1 1 1 1 0 1

Timing: 2 clocks

**CLI=Clear Interrupt**

Flags: O D I T S Z A P C  
          0

1 1 1 1 1 0 1 0

Timing: 2 clocks

**STI=Set Interrupt**

Flags: O D I T S Z A P C  
          1

1 1 1 1 1 0 1 1

Timing: 2 clocks

**HLT=Halt**

Flags: O D I T S Z A P C

1 1 1 1 0 1 0 0

Timing: 2 clocks

**WAIT=Wait**

Flags: O D I T S Z A P C

1 0 0 1 1 0 1 1

**LOCK=Bus Lock  
Prefix**

Timing: 3 clocks

Flags: O D I T S Z A P C

1 1 1 1 0 0 0 0

Timing: 2 clocks

**ESC=Escape (To  
External Device)**

Flags: O D I T S Z A P C

1 1 0 1 1 x x x mod x x x r/m

Timing: 7+EA clocks

#### NOTES:

If d=1 then "to"; if d=0 then "from."

If w=1 then word instruction; if w=0 then byte instruction.

If s:w=01 then 16 bits of immediate data form the operand.

If s:w=11 then an immediate data byte is sign extended to form the 16-bit operand.

If v=0 then "count"=1; if v=1 then "count" in (CL).

X=don't care.

Z is used for some string primitives to compare with ZF FLAG.

AL=8-bit accumulator.

AX=16-bit accumulator.

CX=Count register.

DS=Data segment.

DX=Variable port register.

ES=Extra segment.

Above/below refers to unsigned value.

Greater=more positive signed values.

Less=less positive (more negative) signed values.

See section I.2 for Operand summary.

See section I.4 for Segment Override summary.

### I.6 PROCESSOR RESET REGISTER INITIALIZATION

Flags=0000H (to disable interrupts and single-stepping)

CS=FFFFH (to begin execution at FFFF0H)

IP=0000H

DS=0000H

SS=0000H

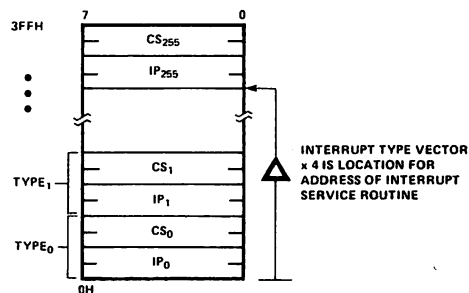
ES=0000H

No other registers are acted upon during reset.

### I.7 8088 RESERVED LOCATIONS

INTERRUPT	LOCATION	FUNCTION
0	00H-03H	Divide by zero
1	04H-07H	Single step
2	08H-0BH	Non-maskable interrupt
3	0CH-0FH	One-byte interrupt instruction
4	10H-13H	Interrupt on overflow

### Interrupt Pointer Table



All mnemonics ©Intel Corporation 1981.

## I.8 8088 INSTRUCTION SET MATRIX

### NOTES:

b=byte operation  
d=direct  
f=from CPU reg  
i=immediate  
ia=immed. to accum.  
id=indirect  
is=immed. byte, sign ext.  
l=long ie. intersegment

m=memory  
r/m=EA is second byte  
si=short intrasegment  
sr=segment register  
t=to CPU reg  
v=variable  
w=word option  
z=zero

## I.9 MNEMONIC INDEX

Mnemonic	Page	Mnemonic	Page	Mnemonic	Page
AAA .....	230	JG .....	240	MOV .....	226
AAD .....	233	JGE .....	240	MOVS .....	236
AAM .....	232	JL .....	239	MUL .....	232
AAS .....	232	JLE .....	239	NEG .....	231
ADC .....	229	JMP .....	237	NOP .....	242
ADD .....	229	JNA .....	239	NOT .....	233
AND .....	235	JNAE .....	239	OR .....	235
CALL .....	237	JNB .....	240	OUT .....	228
CBW .....	233	JNBE .....	240	POP .....	227
CLC .....	242	JNE .....	239	POPF .....	229
CLD .....	242	JNG .....	239	PUSH .....	227
CLI .....	243	JNGE .....	239	PUSHF .....	229
CMC .....	242	JNL .....	240	RCL .....	234
CMP .....	231	JNLE .....	240	RCR .....	234
CMPS .....	236	JNO .....	240	REP .....	236
CWD .....	233	JNP .....	240	RET .....	238
DAA .....	230	JNS .....	240	ROL .....	234
DAS .....	232	JNZ .....	239	ROR .....	234
DEC .....	231	JO .....	239	SAHF .....	228
DIV .....	232	JP .....	239	SAL .....	233
ESC .....	243	JPE .....	239	SAR .....	234
HLT .....	243	JPO .....	240	SBB .....	231
IDIV .....	233	JS .....	239	SCAS .....	237
IMUL .....	232	JZ .....	238	SHL .....	233
IN .....	227	LAHF .....	228	SHR .....	233
INC .....	230	LDS .....	228	STC .....	242
INT .....	241	LEA .....	228	STD .....	243
INTO .....	242	LES .....	228	STI .....	243
IRET .....	242	LOCK .....	243	STOS .....	237
JA .....	240	LODS .....	237	SUB .....	230
JAE .....	240	LOOP .....	240	TEST .....	235
JB .....	239	LOOPE .....	241	WAIT .....	243
JBE .....	239	LOOPNE .....	241	XCHG .....	227
JCXZ .....	241	LOOPNZ .....	241	XLAT .....	228
JE .....	238	LOOPZ .....	241	XOR .....	236

## Appendix J SAMPLE SIRIUS 1 SOFTWARE DRIVERS

PL/M-86 COMPILER      SIRIUS Systems Technology, Inc. (c) 1982    S-1 Hardware      04/01/82      PAGE    1  
Example software drivers for S-1 Hardware

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE HARDWARE  
NO OBJECT MODULE REQUESTED  
COMPILER INVOKED BY: P.86 TEMP.SRC OPTIMIZE(3) PAGELength(42) PAGEWIDTH(109) PRINT(:F4:HW.LS) NOOBJECT

```
$TITLE                                     (c) 1982
$SUBTITLE ('Example software drivers for S-1 Hardware')
/*****
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *****/
```

PL/M-86 COMPILER      Example software drivers for S-1 Hardware      04/01/82      PAGE    2

```
$seject
$SMALL ROM

1      Hardware: do;

2      1      Declare dcl      literally 'declare';
3      1      Dcl      lit      literally 'literally';
4      1      Dcl      addr      lit      'address',
                 ext      lit      'external',
                 init      lit      'initial',
                 intg      lit      'integer',
                 proc      lit      'procedure',
                 ptr      lit      'pointer',
                 pub      lit      'public',
                 rent      lit      'reentrant',
                 ret      lit      'return',
                 struc      lit      'structure',
                 boolean      lit      'byte',
                 true      lit      '0FFH',
                 false      lit      '0000H';
```

PL/M-86 COMPILER      KB: Hardware bit defs      04/01/82      PAGE    3

```
$subtitle('KB: Hardware bit defs')

5      1      dcl SR$intbit      lit '4';      /* KB shift register interrupt mask in 6522 IER/IFR */
6      1      dcl SR$enable      lit '0ch';      /* KB shift register enable in 6522 ACR */
7      1      dcl CBI$intbit      lit '10h';      /* KB RDY edge-sense interrupt mask 6522 PCR */
8      1      dcl CBI$pos_edge      lit '10h';      /* KB RDY edge-sense control in 6522 PCR */

9      1      dcl kb$databit      lit '40h';      /* KB DATA level */
10     1      dcl kb$ackctl      lit '2';      /* KB ACK control for 6522 output */
11     1      dcl kb$TIMEOUT      lit '300';      /* error timeout in milliseconds */
12     1      dcl timer1_ena      lit '0c0h';      /* timer 1 interrupt mask in 6522 IER/IFR */
```

```

$object
/* KYBRD PORT (e8040..e804f) */
13 1      dcl via(16) struc(                                /* 6522 port organization */
                                RB      byte,
                                RA      byte,
                                DDRB    byte,
                                DDRA    byte,
                                TIMER1  word,
                                TIMER1L word,
                                TIMER2  word,
                                SR      byte,
                                ACR      byte,
                                PCR      byte,
                                IFR      byte,
                                IER      byte,
                                RAX      byte) at(0e8000h);
14 1      dcl kb$state      byte;                          /* current state of keyboard stateware */
15 1      dcl kb$data       byte;                          /* constructed data from keyboard */

                                /* nybble convert table for inverted shift reg */
16 1      dcl Ctable(*) byte data (0,8,4,0ch, 2,0ah,6,0eh, 1,9,5,0dh, 3,0bh,7,0fh);
17 1      dcl tick          lit '50';                      /* console clock rate in milliseconds */

```

```

$subtitle('KB: external routines')
/*
 *   signal user about keyboard error state -- ring bell
 */
18 1      dcl signal$KB$error lit 'Ringbell';
        /* Ringbell found in SOUND module */

/*
 *   Process key board event -- in external module
 */
19 1      Process$Event: proc(event) byte ext;
20 2      dcl event byte;
21 2      end;

/*
 *   Software clock resource -- set timeout for interrupt to KB$reset
 */

22 1      set$KB$clock: proc(Period) ext;
23 2      dcl Period intg;                                /* timeout delay in milliseconds */
24 2      end set$KB$clock;

```

```

$subtitle('KB: Keyboard Stateware')
/*
 *   KB interrupt entry (level 6)
 */
25 1      kb$irq: proc pub rent;
26 2      do case kb$state;
        /*
         *   state 0 to state 1: shift register (full) interrupt
         */
27 3      kbst0: do;
28 4          via(4).ACR= via(4).ACR and not SR$enable; /* disable shift register */
                /* prepare for interrupt on negative edge of KB RDY */
29 4          via(4).PCR= via(4).PCR and not CBI$pos_edge;
30 4          via(4).IER= 80h or CBI$intbit;
31 4          disable;
32 4          kb$data = via(4).SR;
                /* time critical section */
                /* get KB data from SR (clears SR IRQ) */

```

```

33 4      via(4).IER= SR$intbit;          /* disable SR interrupt          */
34 4      via(4).RB = via(4).RB or kb$ackctl; /* assert KB ACK control on interrupt */
35 4      enable;                          /* (CBI IRQ is reset)          */
36 4      kb$state = 1;                    /* end of critical section      */
37 4      end;                             /* set to state 1              */

/*
* state 1 to state 2: interrupt from negative edge on KB$RDY
*/
38 3      kbst1: do;
39 4          disable;
40 4          if (via(4).RA and kb$databit) || 0 then /* time critical section      */
41 4              call kb$error; /* if data bit is not low then */
42 4          else do; /* stop bit error has occurred */
43 5              /* prepare for interrupt on positive edge of KB RDY */
44 5              via(4).PCR= via(4).PCR or CBI$pos_edge;
45 5              via(4).RB = via(4).RB and not kb$ackctl; /* release KB ACK control on interrupt */
46 5              kb$state = 2; /* (CBI IRQ is reset)          */
47 4          end; /* set to state 2              */
48 4          enable; /* end of critical section      */
49 4          end;

```

PL/M-86 COMPILER

KB: Keyboard Stateware

04/01/82

PAGE 7

\$eject

```

/*
* state 2 to state 0: interrupt from positive edge on KB$RDY
*/
49 3      kbst2: do;
50 4          if (via(4).RA and kb$databit) = 0 then /* if data bit is low then      */
51 4              call kb$error; /* stop bit error has occurred */
52 4          else do;
53 5              call kb$reset; /* reset hardware/software for next event */
54 5              /* call event processing routine with order of bits reversed to */
55 5              /* reflect physical key number and event type (open or close) */
56 5              if not Process$Event( shl(Ctable(kb$data and 0fh),4)
57 5                  or Ctable(shr(kb$data,4)) ) then /* signal error in event process */
58 3                  end;
59 2      end kbst2;

```

PL/M-86 COMPILER

KB: Keyboard support routines

04/01/82

PAGE 8

\$subtitle('KB: Keyboard support routines')

```

60 1      kb$reset: proc rent; /* puts KB hardware/software into state 0 */
61 2          dcl dummy byte;

62 2          via(4).IER = CBI$intbit; /* clear CBI interrupts          */
63 2          via(4).RB = via(4).RB and not kb$ackctl; /* release kb$ack                */
64 2          via(4).ACR = via(4).ACR or SR$enable; /* enable shift reg              */
65 2          dummy = via(4).SR; /* clr any pending irq           */
66 2          via(4).IER = 80h or SR$intbit; /* enable sr interrupts          */
67 2          kb$state = 0; /* init keybrd state            */
68 2          call set$KB$clock(0); /* clear timeout counter         */
69 2      end kb$reset;

70 1      kb$error: proc rent;
71 2          via(4).RB = via(4).RB or kb$ackctl; /* force kb$ack high            */
72 2          via(4).IER = 7fh; /* allow no interrupts           */
73 2          call set$KB$clock(kb$TIMEOUT); /* time out keyboard            */
74 2      end kb$error;

75 1      kb$init: proc pub rent;

76 2          via(4).RB = via(4).RB and (OFFh-3);
77 2          via(4).DDRA = via(4).DDRA and not kb$databit;
78 2          via(4).DDRB = via(4).DDRB or kb$ackctl;
79 2          via(4).IER = 7fh;
80 2          via(4).PCR = 0;
81 2          via(4).ACR = 0;

```

```

82 2      via(2).ACR= (via(2).ACR and 0c0h) or 40h;
83 2      via(2).timer1L= tick*1000;
84 2      via(2).IER = timer1_ena and 7fh;
85 2      call kb$reset;
86 2      end kb$init;

```

PL/M-86 COMPILER SIRIUS Systems Technology, Inc. (c) 1982 S-1 Hardware 04/01/82 PAGE 9  
 CRTreg: controller chip registers

\$SUBTITLE ('CRTreg: controller chip registers')

```

87 1      DCL CRT$0      byte  AT (0E8000H);          /* CRT-chip address register */
88 1      DCL CRT$1      BYTE  AT (0E8001H);          /* CRT-chip internal register port */

/*
 *      Set CRT register
 */
89 1      set$CRT$reg: proc (reg,value) rent;
90 2          dcl reg byte;
91 2          dcl value byte;
92 2          CRT$0= reg;          /* select register */
93 2          CRT$1= value;        /* set data */
94 2      end set$CRT$reg;

```

PL/M-86 COMPILER CRTreg: cursor-display mode control 04/01/82 PAGE 10

\$SUBTITLE ('CRTreg: cursor-display mode control')

```

95 1      dcl rast$start lit  '10';          /* CRT reg: cursor-start & cursor-display mode */

96 1      DCL Cursor$PAR  BYTE; /* VAR: contents for CRT cursor-start raster & cursor display mode */
97 1      dcl blink$on    boolean;          /* FLAG: =0 Blinking cursor on (fast) */
98 1      dcl curs$off    boolean;          /* FLAG: ||0 Cursor off */

/*
 *      Set cursor to current Cursor parameter byte.
 */
99 1      set$cursor: proc rent;
100 2          call set$CRT$reg(rast$start,Cursor$PAR); /* set raster start reg */
101 2      end set$cursor;

/*
 *      Set block cursor.
 */
102 1      BLOCK$CRS:PROC RENT;
103 2          Cursor$PAR = Cursor$PAR AND 0E0h; /* set block cursor */
104 2          call set$cursor; /* set cursor mode reg */
105 2      END BLOCK$CRS;

/*
 *      Set underscore cursor.
 */
106 1      UNDERSCORE$CRS:PROC RENT;
107 2          Cursor$PAR = 00Fh OR (Cursor$PAR AND 0E0h); /* set underscore cursor */
108 2          call set$cursor; /* set cursor mode reg */
109 2      END UNDERSCORE$CRS;

```

PL/M-86 COMPILER CRTreg: cursor-display mode control 04/01/82 PAGE 11

\$seject

/\*  
 \* Return cursor to previous modes: block or underline, steady or flashing  
 \*/

```

110 1      CURSOR$ON:PROC RENT;
111 2          curs$off= false;          /* reset cursor off flag */
112 2          if blink$on then Cursor$par= Cursor$par or 060h; /* set to flashing mode */
113 2          else Cursor$par= Cursor$par and 01Fh; /* set to steady mode */
114 2          call set$cursor; /* set cursor mode reg */
115 2          END CURSOR$ON;
116 2

```

```

/*
 *      Turn cursor off.
 */
117 1  CURSOR$OFF:PROC RENT;
118 2      curs$off= true; /* set cursor off flag */
119 2      Cursor$PAR = 020h OR (Cursor$PAR AND 01Fh); /* set to off mode */
120 2      call set$cursor; /* set cursor mode reg */
121 2      END CURSOR$OFF;

/*
 *      Set cursor blinking.
 */
122 1  CRS$BLINK$ON:PROC RENT;
123 2      blink$on= true; /* set blinking on flag */
124 2      if not curs$off then Cursor$PAR= 060h OR Cursor$PAR; /* set flashing,if not off */
126 2      call set$cursor; /* set cursor mode reg */
127 2      END CRS$BLINK$ON;

/*
 *      Set cursor steady.
 */
128 1  CRS$BLINK$OFF:PROC RENT;
129 2      blink$on= false; /* reset blinking on flag */
130 2      if not curs$off then Cursor$PAR= 01Fh and Cursor$PAR; /* set steady,if not off */
132 2      call set$cursor; /* set cursor mode reg */
133 2      END CRS$BLINK$OFF;

```

PL/M-86 COMPILER

CRTreg: Cursor positioning

04/01/82

PAGE 12

\$SUBTITLE ('CRTreg: Cursor positioning')

```

134 1  dcl cursaddrH lit '14'; /* CRT reg: MSByte of cursor location word, bits: xx54$3210 */
135 1  dcl cursaddrL lit '15'; /* CRT reg: LSByte of cursor location word */

/*
 *      Position Cursor to Absolute Font Cell number
 *      and display bank
 */
136 1  POS$Cursor: proc (Cell$number) pub rent;
137 2      dcl Cell$Number word; /* Absolute Font Cell Number & diplay bank */
138 2      call set$CRT$reg (cursaddrL, low(Cell$number));
139 2      call set$CRT$reg (cursaddrH, high(Cell$number));
140 2      end POS$Cursor;

```

PL/M-86 COMPILER

CRT: video contrast & brightness

04/01/82

PAGE 13

\$SUBTITLE ('CRT: video contrast & brightness')

```

141 1  DCL CBctrl BYTE AT (0E8040H); /* Contrast & Brightness control register */
/* bits: CCCB$BB-- */

/*
 *      Raise video contrast one level.
 */
142 1  contrast$up: proc rent;
143 2      dcl a byte;
144 2      if (a:= (CBctrl + 20h) and 0E0h) [| 0 then /* add & check upper limit */
145 2          CBctrl= (CBctrl and 01FH) or a; /* set contrast, bits: 765 */
146 2      end contrast$up;

/*
 *      Lower video contrast one level.
 */
147 1  contrast$down: proc rent;
148 2      dcl a byte;
149 2      if (a:= (CBctrl - 20h) and 0E0h) [| 0E0h then /* sub & check lower limit */
150 2          CBctrl= (CBctrl and 01FH) or a; /* set contrast, bits: 765 */
151 2      end contrast$down;

/*
 *      Raise video brightness one level.
 */
152 1  bright$up: proc rent;
153 2      dcl a byte;
154 2      if (a:= (CBctrl + 4) and 01CH) [| 0 then /* add & check upper limit */
155 2          CBctrl= (CBctrl and 0E3H) or a; /* set brightness, bits: 432 */
156 2      end bright$up;

```



```

/*
*      Lower video brightness one level.
*/
157 1  bright$down: proc rent;
158 2  dcl a byte;
159 2  if (a= (CBctrl - 4) and 01Ch) [ 01Ch then      /* sub & check lower limit */
160 2  CBctrl= (CBctrl and 0E3H) or a;                /* set brightness, bits: 432 */
161 2  end bright$down;

```

PL/M-86 COMPILER

CRT: display RAM/Font Cells

04/01/82

PAGE 14

```

$SUBTITLE ('CRT: display RAM/Font Cells')

162 1  dcl screen$ram word at (0F0000h);      /* memory address of display RAM */
163 1  dcl screen$addr ptr;                  /* display ram pointer, base of word ARRAY */
164 1  DCL SCREEN based screen$addr (2000) word; /* ARRAY of Font Cell Pointers */

/*
*      Screen Buffer Word variables
*/

165 1  dcl char$mode      word      pub;      /* CRT attribute bits: 7654$3--- */
166 1  dcl char$base      word      pub;      /* CRT Font Cell Pointer base for */
                                           /* ASCII symbol index */

167 1  DCL REVBIT        LIT      '8000H';
168 1  DCL BGBIT         LIT      '4000H';
169 1  DCL UNDBIT        LIT      '2000H';
170 1  dcl INVBIT        lit      '1000h';
171 1  dcl extraBIT      lit      '0800h';

/*
*      Display symbol from character set (typically ASCII)
*      at absolute Font Cell number
*      (typically: [line] * [display width] + [column] )
*      with current Cursor & Display modes.
*/

172 1  Display$symbol: proc (Symbol$code,Cell$number) pub rent;
173 2  dcl Symbol$code byte;                  /* Symbol print code */
174 2  dcl Cell$Number word;                  /* Absolute Font Cell Number */
175 2  screen(Cell$Number)= (Symbol$code + char$base) OR char$mode;
176 2  end Display$symbol;

```

PL/M-86 COMPILER

CRT hardware initialization

04/01/82

PAGE 15

```

$SUBTITLE ('CRT hardware initialization')

177 1  DCL CRT$config (*) BYTE DATA (92,80, 81,0CFh, 25,6, 25,25, 3,14, 0,15, 0,0, 0,0); /* COMMENT THIS !!!! */

178 1  CRT$Init: PROC;
179 2  DCL I BYTE;

180 2  screen$addr= @screen$ram;

181 2  char$mode= BGBIT;
182 2  char$base= 20;

183 2  curs$off= false;
184 2  blink$on= false;
185 2  Cursor$PAR= 0;

186 2  DO I=0 TO 0FH;
187 3  CALL SET$CRT$REG (I,(CRT$config(I)));
188 3  END;

189 2  END CRT$Init;

```

```
$SUBTITLE ('SOUND variables & hardware defs')
```

```

190 1    dcl bell$freq LIT    '76';                /* period of bell tone: frequency= 14.9KHz */
191 1    dcl    codec$clk      word at (0E8084h);    /* TIMER1: codec clock frequency */
192 1    dcl    codec$ctl      byte at (0E808Bh);    /* ACR: codec clock control register */
193 1    dcl    codec$sda      word at (0E8060h);    /*
194 1    dcl    volume         byte at (0E802Ah);    /* SR: volume shift-register */
195 1    dcl    vol$ctl        byte at (0E802Bh);    /* ACR: SR control register */
196 1    dcl    vol$clk        word at (0E8028h);    /* TIMER2: volume SR clock */

197 1    dcl bell$on byte;                /* FLAG: bell sound presently active */
198 1    dcl vol$level byte;    /*current volume level (nine levels: 0 -- 8)
199 1    dcl vol$table (*) byte data (0FFh,7FH,3FH,1FH,0FH,7,3,1,0);    /* volume shift pattern lookup table */

```

```
$SUBTITLE ('SOUND: Bell control')
```

```

/*
 * Software clock resource -- set timeout for interrupt to Bell$clock
 */

200 1    set$BELL$clock: proc (Period) ext;
201 2    dcl Period intg;                /* timeout delay in milliseconds */
202 2    end set$BELL$clock;

/*
 * CODEC Hardware reset
 */

203 1    Bell$init: proc pub rent;
204 2    vol$level= length(vol$table)-2;    /* set initial volume level near max */
205 2    call Bell$clock;                /* set hardware to a known & quiet state */
206 2    end Bell$init;

```

```
$eject
```

```

207 1    Bell$clock:    proc pub rent;
208 2    codec$ctl = codec$ctl and not 0C0h;    /* disable codec clock */
209 2    codec$sda = 5E00h;    /* initialize codec SDA to input mode... */
210 2    codec$sda = 0D40h;    /* ... to reduce extraneous noise */
211 2    codec$sda = 0AA80h;
212 2    codec$sda = 00C0h;

213 2    vol$ctl = (vol$ctl and not 3Ch) or 10h;    /* set SR & T2 volume register modes */
214 2    vol$clk = 1;    /* volume clock frequency set beyond perception */
215 2    volume = vol$table(vol$level);    /* set volume to current level */
216 2    bell$on = false;    /* set bell state to off */
217 2    end bell$clock;

218 1    Ring$bell: proc pub rent;
219 2    if not bell$on then do;
220 3    call bell$clock;    /* start bell if sound is off */
221 3    codec$sda = 0f80h;    /* init codec hardware on every bell */
222 3    /* set output waveform to 4 up & 4 down, */
223 3    /* a low amplitude triangle wave. */
224 3    codec$ctl = codec$ctl or 0c0h;    /* set codec clock to free run */
225 3    codec$clk = bell$freq;    /* set audio pitch frequency */
226 3    bell$on = true;    /* set bell state on */
227 2    call set$bell$clock(100);    /* turn off bell in 100 milliseconds */
228 2    end;

```

```

$SUBTITLE ('SOUND: volume control')
/*
 *   Raise CODEC volume one level.
 */
229 1  volume$up: proc rent;
230 2      if vol$level |= length(vol$stable)-1 then      /* check upper limit      */
231 2          vol$level= length(vol$stable)-1;          /* set to max volume      */
232 2          else vol$level= vol$level+1;              /* bump level up by one   */
233 2          volume= vol$stable(vol$level);            /* set volume register    */
234 2      end volume$up;

/*
 *   Lower CODEC volume one level.
 */
235 1  volume$down: proc rent;
236 2      if vol$level |= length(vol$stable)-1 then      /* check upper limit      */
237 2          vol$level= length(vol$stable)-2;          /* set to max volume-1    */
238 2          else
239 2              if vol$level[0] then vol$level= vol$level-1; /* drop level by one      */
240 2              volume= vol$stable(vol$level);          /* set volume register    */
241 2      end volume$down;

```

```

$subtitle('SIO: Serial I/O dvrs for TTY: and UL1:')

/*ctr device dcls*/
242 1  dcl sioctr struc
      (adata byte,
       bdata byte,
       xxx byte,
       ctrctl byte) at (0E0020h);
/*sio device dcls*/
243 1  dcl siodev struc
      (adata byte,
       bdata byte,
       actl byte,
       bctl byte) at(0E0040h);
244 1  dcl rx$avail literally '1',
      tx$empty literally '4';

245 1  dcl serial_params struc
      (actrlsb byte,          /*LSByte of chan a.'s baud rate  */
       actrmsb byte,          /*MSByte ...                      */
       bctrslb byte,          /*LSByte of chan b.'s baud rate  */
       bctrmsb byte,          /*MSByte ...                      */
       /* if [baud] then lsb = ??h msb = ??h 1.25Mhz/([baud]*16)
          50 ===| 1Ah      06h      50.00 -0- (min.tol.dist.43.75%)
          75 ===| 11h      04h      75.00 -0- ( " 43.75%)
          110 ===| C6h      02h      110.00 -0- ( " 43.75%)
          134.5 ===| 44h      02h      134.00 -0.37% ( " 40.23%)
          150 ===| 08h      02h      150.00 -0- ( " 43.75%)
          200 ===| 86h      01h      200.00 -0- ( " 43.75%)

          300 ===| 04h      01h      300.00 -0- ( " 43.75%)
          600 ===| 82h      00h      600.00 -0- ( " 43.75%)
          1.2k ===| 41h      00h      1201.00 +0.08% ( " 42.99%)

```

\$eject

	2Ch	00h	1775.00	-1.39%	(	"	30.54%)
1.8k ===	2Bh	00h	1816.00	+0.09%	(	"	42.88%)
	28h	00h	1953.00	-2.36%	(	"	21.33)
2.0k ===	27h	00h	2003.00	+0.15%	(	"	42.32)
	21h	00h	2367.00	-1.38%	(	"	30.64%)
2.4k ===	20h	00h	2441.00	+1.71%	(	"	27.51%)

3.6k ===	16h	00h	3551.00	-1.36%	(	"	30.83%)
	15h	00h	3720.00	+3.33%	(	"	12.4%)
4.8k ===	11h	00h	4595.00	-4.27%	(	"	3.185%)
	10h	00h	4882.00	+1.02%	(	"	34.06%)
9.6k ===	09h	00h	8680.55	-9.58%	(DISTORTED)		
	08h	00h	9765.56	+1.73%	(min.tol.dist.27.32%)		
	06h	00h	13020.83	-9.58%	(DISTORTED)		
	05h	00h	15625.00	+8.51%	(DISTORTED)		
19.2k ===	05h	00h	15625.00	-18.62%	of 19.2k (DISTORTED)		
	04h	00h	19531.25	+1.02%	(min.tol.dist.34.06%)		

min.tol.dist. figure assumes no channel noise effects.  
NOTE: possible noise DOES NOT includes bias distortion  
caused by various cable capacitance effects\*/

PL/M-86 COMPILER

SIO: Serial I/O dvr's for TTY: and UL1:

04/01/82

PAGE 22

\$seject

```

cr2a  byte,      /*bus interface option: 10h if baud a [= baud b
                                14h if baud a | baud b*/

cr4a  byte,
cr4b  byte,
/*cr4x (16x)$54$(stops)$(even)$(parenb) = 4?h
   01      00 ss      e      p
           ss = 01 1 stop
               = 10 1.5 stop
               = 11 2 stop
               e = 1 even
               e = 0 odd, byte transparent
               p = 1 even or odd
               p = 0 byte transparent*/

cr3a  byte,
cr3b  byte,
/*cr3x (rbits)$(autoenb)$4$3$2$1$(renb) = ?1h
   bb      1      0 0 0 0 1
   bb = 11 byte transparent cr3x = E1h
       = 01 even,odd        cr3x = 61h*/

cr5a  byte,
cr5b  byte) EXT;
/*cr5x (dtr)$(tbits)$(br)$(tenb)$2$(rts)$0 = ?Ah
   1      bb      0      1      0      1      0
   bb = 11 space,mark cr5x = EAh
       bb = 01 even,odd,no cr5x = AAh*/

```

PL/M-86 COMPILER

SIO: Serial I/O dvr's for port A -- TTY\$INSTAT & TTY\$STAT

04/01/82

PAGE 23

\$subtitle('SIO: Serial I/O dvr's for port A -- TTY\$INSTAT & TTY\$STAT')

```

246  1  TTY$in$stat:proc boolean PUB;
247  2  if ( (siodev.act1 AND rx$avail) [| 0)
249  2  then return(true);
249  2  return(false);
250  2  end TTY$in$stat;

251  1  TTY$stat:proc boolean PUB;
252  2  if ( (siodev.act1 AND tx$empty) = 0)
254  2  then return(true);
254  2  return(false);
255  2  end TTY$stat;

```

```
$subtitle('SIO: Serial I/O dvr's for port A -- TTY$GET & TTY$PUT')
```

```
256 1  TTY$get:proc byte PUB;
      /*user must not activate this procedure if siodev chan. a reg. ptr
      is not set to 0 (only [] 0 if user has been mucking with hardware*/
257 2  do while( (siodev.actl AND rx$avail) = 0);      /*wait forever till empty */
258 3  end;
259 2  return(siodev.adata);      /*input form 7201 */

260 2  end TTY$get;

261 1  TTY$put:proc(char) PUB;
262 2  dcl char byte;

      /*user must not activate this procedure if siodev chan. a reg. ptr
      is not set to 0 (only [] 0 if user has been mucking with hardware*/
263 2  do while( (siodev.actl AND tx$empty) = 0);      /*wait forever till empty */
264 3  end;
265 2  siodev.adata = char;      /*output a char */
266 2  return;
267 2  end TTY$put;
```

```
$subtitle('SIO: Serial I/O dvr's for port B -- UL1$STAT & UL1$PUT')
```

```
268 1  UL1$stat:proc boolean PUB;
269 2  if ( (siodev.bctl AND tx$empty) = 0)
271 2  then return(true);
      return(false);
272 2  end UL1$stat;

273 1  UL1$put:proc(char) PUB;
274 2  dcl char byte;

      /*user must not activate this procedure if siodev chan. b reg. ptr
      is not set to 0 (only [] 0 if user has been mucking with hardware*/
275 2  do while( (siodev.bctl AND tx$empty) = 0);      /*wait forever till empty */
276 3  end;
277 2  siodev.bdata = char;      /*output a char */
278 2  return;
279 2  end UL1$put;
```

```
$subtitle('SIO: Serial I/O dvr's for ports A & B -- SIO$INIT')
```

```
280 1  SIO$init:proc PUB;
281 2  siodev.actl = 00$011$000b;      /*chan. a reset */
282 2  siodev.bctl = 00$011$000b;      /*chan. b reset */

      /*load timer now; cant touch 7201 chip for 4 2.5Mhz clocks*/
283 2  siocr.ctrctl = 36h;      /*7$(ctra)$ (rl)$ (mode)$ (bin) */
284 2  siocr.adata = serial_params.actrlsb;
285 2  siocr.adata = serial_params.actrmsb;
286 2  siocr.ctrctl = 76h;      /*7$(ctrb)$ (rl)$ (mode)$ (bin) */
287 2  siocr.bdata = serial_params.bctrslb;
288 2  siocr.bdata = serial_params.bctrmsb;

      /*cr2a bus interface option*/
289 2  siodev.actl = 2;      /*---|cr4a */
290 2  siodev.actl = serial_params.cr2a;

      /*cr4x*/
```

```

291 2      siodev.act1 = 4;                      /*--|cr4a          */
292 2      siodev.act1 = serial_params.cr4a;
293 2      siodev.bctl = 4;                      /*--|cr4b          */
294 2      siodev.bctl = serial_params.cr4b;

/*cr3x*/
295 2      siodev.act1 = 3;                      /*--|cr3a          */
296 2      siodev.act1 = serial_params.cr3a;
297 2      siodev.bctl = 3;                      /*--|cr3b          */
298 2      siodev.bctl = serial_params.cr3b;

```

PL/M-86 COMPILER

SIO: Serial I/O dvr's for ports A & B -- SIO\$INIT

04/01/82

PAGE 27

```

$seject

/*cr5x*/
299 2      siodev.act1 = 5;                      /*--|cr5a          */
300 2      siodev.act1 = serial_params.cr5a;
301 2      siodev.bctl = 5;                      /*--|cr5b          */
302 2      siodev.bctl = serial_params.cr5b;

/*cr0x reset ext/st intrs to enable modem control sense--| autoenb chans.
also --| crlx, set intr params*/
303 2      siodev.act1 = 00$010$001b;
304 2      siodev.act1 = 0;                      /*no intrs         */
305 2      siodev.bctl = 00$010$001b;
306 2      siodev.bctl = 0;                      /*no intrs         */
307 2      end sio$init;

```

PL/M-86 COMPILER

PPORT -- centronics interface routines

04/01/82

PAGE 28

```

$subtitle ('PPORT -- centronics interface routines ')

/*
 * This module implements the initialization, LISTST, and LIST functions
 * for a Centronics-compatible parallel printer interface, using the
 * 6522 VIA chip.
 *
 * Our entry points are named pp$init, LPT$stat, and LPT$put respectively,
 * it's up to our caller to decode the I/O byte and call the approp-
 * riate routines.
 */

```

PL/M-86 COMPILER

PPORT -- centronics interface routines

04/01/82

PAGE 29

```

$seject
308 1      declare pp$base pointer;
309 1      declare pp based pp$base structure (
        rb byte,
        ra byte,
        ddrb byte,
        ddra byte,
        tlcl byte,
        tlch byte,
        tlll byte,
        tllh byte,
        t2cl byte,
        t2ch byte,
        sr byte,
        acr byte,
        pcr byte,
        ifr byte,
        ier byte,
        rax byte
        );
/*
 * Bit definitions for Centronics-style parallel interface, 'vial'.
 */
310 1      declare vial$base literally '0e8020h';
311 1      declare ds$1 literally '01h';
312 1      declare pi$h literally '02h';
313 1      declare bz$h literally '20h';

/* baseaddr for a 6522
   6522 template
   out-in reg 'b'
   out-in reg 'a'
   data-direction, reg 'b'
   data-direction, reg 'a'
   t1 ctr(r)/lat(w) lo
   t1 ctr hi
   t1 latch lo
   t1 latch hi
   t2 ctr(r)/lat(w) lo
   t2 ctr hi
   shift register
   auxiliary ctrl reg
   peripheral ctrl reg
   interrupt flg register
   interrupt enbl register
   out-in reg 'a' NO HANDSHAKE
*/

```

```

314 1 declare ak$1 literally '40h';          /* printer ack (pb6)          */
315 1 declare sl$h literally '80h';          /* on-line and no error (pb7) */
/*
/* Bit definitions for multi-use pio, 'via2'.
*/
316 1 declare via2$base literally '0e8040h'; /* baseaddr for this chip    */
317 1 declare te$h literally '0lh';          /* talk-enable line          */

```

PL/M-86 COMPILER PPORT -- centronics interface routines 04/01/82 PAGE 30

```

$seject
/*
* initial setup for parallel printer port
* Note we use via2 during this setup to get talk-enable turned on, and
* thus someone MUST ALREADY HAVE VIA2 INITIALIZED.
*/
318 1 pp$init: procedure public;
319 2   pp$base = via2$base;          /* point to secondary chip for te */
320 2   pp.rb = pp.rb or te$h;        /* set 'talk enbl'              */
321 2   pp$base = vial$base;          /* point struc at primary chip   */
322 2   pp.ra = 0;                    /* ra is dataport, init with 0's */
323 2   pp.ddra = 0ffh;              /* set all ra bits as outgoing   */
324 2   pp.rb = ds$1;                /* rb is ctrlport, init no ds/pi */
325 2   pp.ddrb = ds$1 or pi$h;      /* these 2 only are outgoing     */
/* cal/ca2 cbl/cb2 not used      */
/* timers/shiftreg not used      */
326 2 end pp$init;

```

PL/M-86 COMPILER SIRIUS Systems Technology, Inc. (c) 1982 S-1 Hardware 04/01/82 PAGE 31

```

$seject
/*
* Test status of printer, return true if on-line and not busy, else
* false. For some reason, the Altos code explicitly deasserted data
* strobe before testing; we'll assume that this represents an Altos
* fubar and is not required here.
*/
327 1 LPT$stat: procedure byte public;
328 2   if (pp.rb and (sl$h or bz$h)) = sl$h then return 0ffh;
329 2   return 0;
330 2 end LPT$stat;
/*
* Put one character to the printer interface.
*/
332 1 LPT$put: procedure(ch) public;
333 2   declare ch byte;
334 2   do while LPT$stat = 0; end;
335 2   pp.ra = ch;          /* wait for printer ready      */
336 2   pp.ra = ch;          /* put outgoing char on the port */
337 2   disable;
338 2   pp.rb = pp.rb and not ds$1; /* assert data strobe          */
339 2   pp.rb = pp.rb or ds$1;     /* deassert data strobe        */
340 2   enable;
341 2   return;
342 2 end LPT$put;

```

```
        $SUBTITLE ('Example software drivers for S-1 Hardware')  
343  1    end Hardware;
```

## MODULE INFORMATION:

```
CODE AREA SIZE    = 073EH    1854D  
CONSTANT AREA SIZE = 0000H     0D  
VARIABLE AREA SIZE = 0014H    20D  
MAXIMUM STACK SIZE = 000EH    14D  
807 LINES READ  
0 PROGRAM WARNINGS  
0 PROGRAM ERRORS
```

END OF PL/M-86 COMPILATION



---

## INDEX

<b>A</b>	Addition .....	66
	Address generation .....	13
	Addressing modes .....	28, 29
	Addressing structures .....	31
	Arbitrator circuit .....	39
	Arithmetic instructions .....	63
	Assembly language .....	223
	Attribute bits .....	40, 41
	Audio amplifier .....	37
	Audio clock .....	36
	Audio hardware .....	109
	Audio section .....	36
	Auxiliary carry flag .....	8
<b>B</b>	Base address .....	11
	Based addressing .....	31
	Based indexed addressing .....	33
	Bit clock .....	36
	Bit manipulation instructions .....	69
	Bit-mapped display .....	44
	Bit shift .....	48
	Boolean operators .....	69
	Boot ROM .....	35
	Breakpoint interrupt .....	27
	Brightness .....	43
	Brightness and contrast control .....	97
	Bus control logic .....	147
	Bus interface unit .....	4, 5
	Byte ready .....	19
	Byte-ready strobe .....	45
<b>C</b>	Carry flag .....	8
	Centronics interface .....	1, 36
	Checksum .....	45, 46, 50
	CLI (clear interrupt-enable) .....	21
	Clock and reset control logic .....	154
	Clock recovery .....	45, 49
	Coder/Decoder (CODEC) .....	36, 109, 122
	Companding .....	38, 123
	Conditional Transfers .....	77
	Continually-Variable-Slope Delta (CVSD) modulation .....	109
	Contrast .....	43
	Control port .....	1, 36
	Control register 0 .....	155
	Control register 1 .....	115, 121, 157
	Control register 2 .....	115, 121
	Control register 2 (channel A) .....	159
	Control register 2 (channel B) .....	161
	Control register 3 .....	116, 121, 161
	Control register 4 .....	163
	Control register 5 .....	164
	Control register 6 .....	167

Control register 7 .....	167
CPU (central processing unit) .....	3
CRTC device operation .....	99
CS register .....	7
Cursur control .....	107
<b>D</b> Data block .....	45
Data block ID .....	45
Data bytes .....	50
Data field .....	50
Data ID .....	50
Data register .....	6
Data sync .....	50
Data transfer .....	45
Data transfer instructions .....	58
Delta modulation .....	39, 122
Digital recording .....	36
Direct addressing .....	30
Direct memory access .....	19
Direction flag .....	8
Disk drive assembly .....	3, 44
Disk drive interface .....	45
Disk interface .....	38
Display .....	39
Display circuit .....	97
Display contrast .....	43
Display system .....	95
Display unit .....	53
DIV (division) .....	22
Division .....	68
DMA control logic .....	152
Double words .....	11
DS register .....	7
Dual port memory .....	39
Dynamic relocation .....	16
<b>E</b> Effective address .....	29
8048 .....	49
8088 instruction set .....	245
8088 register .....	223
8080/8085 .....	9
8253 timer chip .....	36
ES register .....	7
Execution unit .....	4, 5
External synchronization .....	81
External interrupts .....	21
Expansion bus .....	39, 83
<b>F</b> Fan .....	51
Fetch overlap .....	4
FIFO .....	37
Flag operations .....	80
Flags .....	8, 65
Fold-back limiting .....	51
Font cells .....	41

Font cell address .....	41
Font cells .....	40, 42
Font pointer .....	40
Formatting .....	46
Fuse .....	51
<b>G</b> Gap 1 .....	50
Gap 2 .....	50
GCR (group code recording) .....	49
GCR read circuit .....	45
General register .....	6
<b>H</b> Header ID .....	50
Header search .....	45
Header sync .....	50
Head positioning .....	47
High/Low intensity .....	41
High resolution mode .....	43, 97
HLT (halt) .....	28
Hold (HOLD) .....	19
Hold acknowledge (HLDA) .....	19
<b>I</b> IDIV (integer divide) .....	22
IEEE 488 .....	1, 36
Indexed addressing .....	32
Index registers .....	6
Input/Output (I/O) functions .....	35
Instruction pointer .....	8
Instruction set .....	57
INT 3 (breakpoint interrupt) .....	27
Interface signals to CPU .....	99
Interface signals to display circuits .....	100
Interlace .....	53, 106
Interlace sync mode display .....	106
Interlace sync and video mode display .....	106
Internal interrupts .....	22
Internal registers .....	101
Interrupt control logic .....	148
Interrupt-enable flag .....	9
Interrupt instructions .....	78
Interrupt nesting .....	25
Interrupt pointer table .....	23
Interrupt procedures .....	23, 25
Interrupt request .....	21
Interrupts .....	19, 21, 25
INTO (interrupt on overflow) .....	22
INTR .....	21
I/O address assignments .....	89
I/O port addressing .....	35
IRET (interrupt return) .....	21, 26
Iteration control .....	78

<b>K</b>	Keyboard .....	125
	Keyboard electrical specifications .....	125
	Keyboard interface .....	38
	Keyboard mechanical specifications .....	125
	Keyboard unit .....	55
<b>L</b>	Light pen .....	36
	Line filters .....	51
	LOCK (lock) .....	21
	Logical address .....	13
	Logical instructions .....	69
	Low pass filter .....	38
<b>M</b>	Main logic board .....	3
	Memory .....	10
	Memory access .....	18
	Memory-mapped I/O .....	18
	Modem .....	36
	Motor speed control .....	47
	Motor speed variation .....	47
	MPSC <sup>2</sup> .....	129, 137
	MPSC <sup>2</sup> asynchronous mode .....	135, 139, 144
	MPSC <sup>2</sup> application hints .....	193
	MPSC <sup>2</sup> COP synchronous mode .....	140, 145
	MPSC <sup>2</sup> pin description .....	130
	MPSC <sup>2</sup> receiver .....	142
	MPSC <sup>2</sup> registers .....	154
	MPSC <sup>2</sup> SDLC (/HDLC BOP synchronous) mode .....	141, 146
	MPX <sup>2</sup> synchronous bit-oriented protocols .....	135
	MPSC <sup>2</sup> transmitter .....	137
	Multiplication .....	67
<b>N</b>	NMI (nonmaskable interrupt) .....	22, 25
	Nondisplay .....	41
	Noninterface mode display .....	106
	Nonmaskable interrupt .....	21
<b>O</b>	Offset value .....	11
	Overflow flag .....	9, 21
<b>P</b>	Packed decimal number .....	64
	Parallel port .....	1
	Parity .....	117
	Parity flag .....	8
	Phase-locked loop (PLL) .....	49
	Physical address .....	13, 29
	Pointers .....	6, 11
	Power supply .....	3, 51
	Power switch .....	51
	Priority order .....	23

Processor control instructions .....	79
Processor halt .....	28
Processor-initiated interrupts .....	25
Processor unit .....	3
Program transfer instructions .....	74
Programmable interrupt controller .....	21

## R

Reading data .....	45
Read channel .....	49
Read signal amplitude .....	47
Read/Write head .....	49
Receive data first-in first-out register .....	118
Recording density .....	47
Register indirect addressing .....	31
Register operands .....	26
Repeat .....	21
Reserved memory locations .....	18
Reverse video .....	41
Rotates .....	70
Rotational period .....	51
RS-232 (V-24) .....	1, 36

## S

Screen buffer .....	40
Screen buffer words .....	96
Sector components .....	50
Sector format .....	50
Sector header .....	45, 50
Sector ID .....	50
Sector number .....	45
Segmentation .....	12
Segment override .....	15, 21
Segment registers .....	7
Serial ports .....	36
7201 communications controller .....	129
Shifts .....	70
Signed binary numbers .....	64
Sign flag .....	9
Single-step mode .....	26
6522 versatile interface .....	199
6522 versatile interface electrical characteristics .....	200
6522 versatile interface functional description .....	200
6522 versatile interface peripheral interface characteristics .....	203
6522 versatile interface pins .....	206
6522 versatile interface read timing characteristics .....	201
6522 versatile interface write timing characteristics .....	202
Software attribute .....	41
Software-initiated interrupt .....	26
Sound output .....	36
Sound quality .....	37
Speaker .....	36
Speed control .....	47
Speed control processor (SCP) .....	47
SSDA (synchronous serial data adapter) .....	36, 109
SSDA interface signals for CPU .....	112

SSDA operation .....	110
SSDA registers .....	114
SS register .....	7
Stack pointer .....	17
Stacks .....	17
Stack segment .....	17
Status register .....	119, 120
Status register 0 .....	168
Status register 1 .....	170
Status register 2 .....	172
STI .....	21, 25
Storage organization .....	10
Strikeover .....	41
Strings .....	15
String addressing .....	34
Strings .....	15
Subtraction .....	66
Supervisor call .....	26
Swivel ramp .....	53
SYN .....	45
Sync-code register .....	117
Sync detection .....	45
System reset .....	27

## T

Test (TEST) .....	19
Text mode .....	39
Tones .....	36
Track format .....	51
Track ID .....	50
Track numbers .....	45
Trap flag .....	9, 22, 26
Trim erase .....	46
Type 0 interrupt .....	22
Type 1 interrupt .....	22, 26
Type 3 interrupt .....	27

## U

Unconditional transfers .....	76
Underline/strikeover .....	41
Unpacked decimal numbers .....	64
Unsigned binary numbers .....	64

## V

Verification .....	46
Voice .....	36
Volume control .....	36
Volume level .....	36

## W

Wait (WAIT) .....	19, 45
Word data .....	11
Wrap around .....	14
Write channel .....	50
Writing data .....	46

## Z

Zero flag .....	9
Zones .....	48, 51

Supplemental Technical Reference Material

**SUPPLEMENTAL TECHNICAL REFERENCE MATERIAL**

APPLICATION NOTE: 002

Revision 0

Supplemental Technical Reference Material

**COPYRIGHT**

© 1983 by VICTOR. (R)

All rights reserved. This publication contains proprietary information which is protected by this copyright. No part of this publication may be reproduced, transcribed, stored in a retrieval system, translated into any language or computer language, or transmitted in any form whatsoever without the prior written consent of the publisher.

For information contact:

VICTOR Publications  
380 El Pueblo Road  
Scotts Valley, CA 95066  
(408) 438-6680

**TRADEMARKS**

VICTOR is a registered trademark of Victor Technologies, Inc.

**NOTICE**

VICTOR makes no representations or warranties of any kind whatsoever with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. VICTOR shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this publication or its contents.

VICTOR reserves the right to revise this publication from time to time and to make changes in the content hereof without obligation to notify any person of such revision or changes.

First VICTOR printing March 1983.



**CONTENTS**

	<b>Page</b>	<b>Rev</b>
<b>1. Victor 9000 System Overview</b>		
1.1 Computer .....	1-1	0
1.2 Memory .....	1-1	0
1.3 Disk System .....	1-2	0
1.4 Display System .....	1-3	0
1.5 Keyboard .....	1-4	0
1.6 Memory Map .....	1-5	0
1.6.1 MS-DOS .....	1-6	0
1.6.2 CP/M-86 .....	1-7	0
 <b>2. Display Driver Specifications</b>		
2.1 Overview .....	2-1	0
2.2 Screen Control Sequences .....	2-2	0
2.3 Multi-Character Escape Sequences .....	2-3	0
2.3.1 Cursor Functions .....	2-3	0
2.3.2 Editing Functions .....	2-4	0
2.3.3 Configuration Functions .....	2-6	0
2.3.4 Operation Mode Functions .....	2-7	0
2.3.5 Special Functions .....	2-8	0
2.4 Direct Cursor Addressing - Examples ....	2-10	0
2.4.1 Microsoft MS-BASIC .....	2-10	0
2.4.2 Microsoft MACRO-86 .....	2-11	0
2.4.3 Microsoft MS-Pascal .....	2-12	0
2.5 Transmit Page - Examples .....	2-13	0
2.5.1 Microsoft MS-BASIC .....	2-13	0
2.5.2 Microsoft MACRO-86 .....	2-14	0
2.5.3 Microsoft MS-Pascal .....	2-15	0
 <b>3. Input/Output Port Specifications</b>		
3.1 Device Connection .....	3-1	0
3.2 Parallel Printer Connection .....	3-2	0
3.3 Parallel Cable Requirements .....	3-2	0
3.4 Serial Printer Connection .....	3-3	0
3.5 Serial Cable Requirements .....	3-4	0
3.6 Operating System Port Utilities .....	3-5	0
3.6.1 SETIO - List Device Selection ...	3-5	0
3.6.2 STAT - List Device Selection ....	3-5	0
3.6.3 PORTSET - Baud Rate Selection ...	3-6	0
3.6.4 PORTCONF - Baud Rate Selection ..	3-6	0
3.7 Serial Input/Output Ports .....	3-7	0
3.8 Baud Rate / Transmission - Examples ...	3-8	0
3.8.1 Microsoft MS-BASIC .....	3-9	0
3.8.2 Microsoft MACRO-86 .....	3-11	0

**CONTENTS  
continued**

<b>Appendices</b>	<b>Page</b>	<b>Rev</b>
<b>Appendix A: ASCII Codes</b>		
A.1 ASCII Codes used in the Victor 9000	A-1	0
A.2 ASCII/Hex/Decimal Chart .....	A-2	0
<b>Appendix B: Keyboard</b>		
B.1 Victor 9000 Keyboard Layout .....	B-1	0
<b>Appendix C: Input/Output Ports</b>		
C.1 Parallel (Centronics) Port .....	C-1	0
C.2 Serial (RS232C) Port .....	C-2	0
C.3 IEEE-488 Port .....	C-3	0
C.4 Control Port .....	C-4	0
<b>Appendix D: Assembler Examples</b>		
D.1 MACRO-86 Assembler Shell .....	D-1	0
D.2 ASM-86 Assembler Shell .....	D-2	0
<b>Appendix E: File Header Structure</b>		
E.1 EXE File Header Structure .....	E-1	0
<b>Appendix F: Victor 9000 Specifications</b>		
F.1 Technical Specifications .....	F-1	0
F.2 Physical Specifications .....	F-2	0
<b>Appendix G: Glossary</b>		
G.1 Glossary of Terms .....	G-1	0
<b>Appendix H: MS-DOS Base Page Structure</b>		
H.1 Base Page Structure .....	H-1	0
<b>Appendix I: Interrupt Driven Serial I/O</b>		
I.1 Interrupt Driven Serial I/O .....	I-1	0
I.2 Interrupt Vectors .....	I-2	0
I.2.1 Vectors Available on Victor .	I-2	0
I.2.2 Location of Vectors .....	I-2	0
I.2.3 Set Vector - Example .....	I-3	0
I.3 Enabling Internal/External Clocks .	I-4	0
I.3.1 Providing Clocks .....	I-4	0
I.4 Initializing the SIO .....	I-5	0
I.4.1 Baud Rate for SIO .....	I-6	0
I.4.2 Enabling SIO Interrupts .....	I-6	0
I.5 Interrupt Service Routine (ISR) ...	I-7	0
I.5.1 Interrupt Service - Example .	I-8	0
I.6 Setting Direction Bits .....	I-10	0